SLAM temporel à contraintes multiples

Datta Ramadasan

N° d'ordre : D.U : 2653
EDSPIC : 735

Université Blaise Pascal - Clermont II

Ecole Doctorale
Sciences Pour L'Ingénieur De Clermont-Ferrand

Thèse

présentée par :

Datta RAMADASAN

pour obtenir le grade de

Docteur d'Université

Spécialité : Vision pour la robotique

SLAM temporel à contraintes multiples

Soutenue publiquement le 15 décembre 2015 devant le jury :

M. Lionel LACASSAGNE	Président du jury
M. Cédric DEMONCEAUX	Rapporteur
M. Éric MARCHAND	Rapporteur
M. Michel DHOME	Examinateur
M. Sébastien GOUDE	Invité
M. Marc CHEVALDONNÉ	Encadrant
M. Thierry CHATEAU	Directeur de thèse

Remerciements

Je remercie tout d'abord la région Auvergne et le Fond Européen de développement régional (FEDER) pour avoir financé ce travail de thèse.

Je remercie Cédric Demonceaux, Eric Marchand, Lionel Lacassagne, Michel Dhome et Sébastien Goude pour avoir accepté de participer au jury de thèse et pour avoir souligné les efforts apportés à la réalisation de ce travail.

Je remercie tout particulièrement mes encadrants, Thierry Chateau et Marc Chevaldonné pour leur constante bonne heure, leur soutien et la confiance qu'ils m'ont apporté.

Je remercie les membres de l'Institut Pascal avec lesquels j'ai eu l'occasion d'échanger, pour m'avoir soutenu et encouragé. Je remercie le service technique pour ne pas m'avoir trop sollicité sur le fonctionnement de code antique et pour m'avoir fourni le matériel et les logiciels nécessaires à mes travaux.

Je remercie Pierre Lébraly pour m'avoir montré la voie de l'optimisation à paramètres et contraintes multiples. Je remercie Clément Deymier pour m'avoir montré comment trouver sans chercher et pour ses contributions pertinentes et incomparables à toute la géométrie nécessaire au SLAM.

Je remercie mes collègues de bureau, de couloir, d'équipe et d'axe pour tous les moments partagés.

Enfin, je remercie ma famille, sans oublier Mango et Caillou, pour avoir accepté et compris mon investissement dans ce travail.

Résumé

Ce mémoire décrit mes travaux de thèse de doctorat menés au sein de l'équipe ComSee (*Computers that See*) rattachée à l'axe ISPR (Image, Systèmes de Perception et Robotique) de l'Institut Pascal. Celle-ci a été financée par la Région Auvergne et le Fonds Européen de Développement Régional. Les travaux présentés s'inscrivent dans le cadre d'applications de localisation pour la robotique mobile et la Réalité Augmentée.

Le *framework* réalisé au cours de cette thèse est une approche générique pour l'implémentation d'applications de SLAM : *Simultaneous Localization And Mapping* (algorithme de localisation par rapport à un modèle simultanément reconstruit). L'approche intègre une multitude de contraintes dans les processus de localisation et de reconstruction. Ces contraintes proviennent de données capteurs mais également d'*a priori* liés au contexte applicatif. Chaque contrainte est utilisée au sein d'un même algorithme d'optimisation afin d'améliorer l'estimation du mouvement ainsi que la précision du modèle reconstruit.

Trois problèmes ont été abordés au cours de ce travail. Le premier concerne l'utilisation de contraintes sur le modèle reconstruit pour l'estimation précise d'objets 3D partiellement connus et présents dans l'environnement. La seconde problématique traite de la fusion de données multi-capteurs, donc hétérogènes et asynchrones, en utilisant un unique algorithme d'optimisation. La dernière problématique concerne la génération automatique et efficace d'algorithmes d'optimisation à contraintes multiples. L'objectif est de proposer une solution temps réel [1] aux problèmes de SLAM à contraintes multiples.

Une approche générique est utilisée pour concevoir le *framework* afin de gérer une multitude de configurations liées aux différentes contraintes des problèmes de SLAM. Un intérêt tout particulier a été porté à la faible consommation de ressources (mémoire et CPU) tout en conservant une grande portabilité. De plus, la méta-programmation est utilisée pour générer automatiquement et spécifiquement les parties les plus complexes du code en fonction du problème à résoudre. La bibliothèque d'optimisation LMA qui a été développée au cours de cette thèse est mise à disposition de la communauté en *open-source*.

Des expérimentations sont présentées à la fois sur des données de synthèse et des données réelles. Un comparatif exhaustif met en évidence les performances de la bibliothèque LMA face aux alternatives les plus utilisées de l'état de l'art. De plus, le *framework* de SLAM est utilisé sur des problèmes impliquant une difficulté et une quantité de contraintes croissantes. Les applications de robotique mobile et de Réalité Augmentée mettent en évidence des performances temps réel et un niveau de précision qui croît avec le nombre de contraintes utilisées.

Mots-clés : Reconstruction 3D, SLAM visuel, SLAM contraint, temps réel, ajustement de faisceaux, réalité augmentée, navigation autonome, Levenberg-Marquardt, C++, méta-programmation.

1. Capacité à traiter en ligne l'intégralité des données capteurs.

Abstract

Title : Multiple Constraints and temporal SLAM

This report describes my thesis work conducted within the ComSee (*Computers That See*) team related to the ISPR axis (ImageS, Perception Systems and Robotics) of *Institut Pascal*. It was financed by the Auvergne *Région* and the European Fund of Regional Development. The thesis was motivated by localization issues related to Augmented Reality and autonomous navigation.

The framework developed during this thesis is a generic approach to implement SLAM algorithms : Simultaneous Localization And Mapping. The proposed approach use multiple constraints in the localization and mapping processes. Those constraints come from sensors data and also from knowledge given by the application context. Each constraint is used into one optimization algorithm in order to improve the estimation of the motion and the accuracy of the map.

Three problems have been tackled. The first deals with constraints on the map to accurately estimate the pose of 3D objects partially known in the environment. The second problem is about merging multiple heterogeneous and asynchronous data coming from different sensors using an optimization algorithm. The last problem is to write an efficient and real-time implementation of the SLAM problem using multiple constraints.

A generic approach is used to design the framework and to generate different configurations, according to the constraints, of each SLAM problem. A particular interest has been put in the low computational requirement (in term of memory and CPU) while offering a high portability. Moreover, meta-programming techniques have been used to automatically and specifically generate the more complex parts of the code according to the given problem. The optimization library LMA, developed during this thesis, is made available of the community in open-source.

Several experiments were done on synthesis and real data. An exhaustive benchmark shows the performances of the LMA library compared to the most used alternatives of the state of the art. Moreover, the SLAM framework is used on different problems with an increasing difficulty and amount of constraints. Augmented Reality and autonomous navigation applications show the good performances and accuracies in multiple constraints context.

Keywords : Structure from motion, Visual SLAM, Constraint SLAM, Real-Time, Bundle Adjustment, Augmented Reality, Autonomous Navigation, Levenberg-Marquardt, C++, meta-programming.

Table des matières

Liste des figures

Liste des tableaux

Notations

R	Matrice de rotation.
t	Vecteur de translation.
$\mathcal{C}$	Ensemble de poses caméra.
$\mathcal{P}$	Ensemble de points 3D.
P_w	Point 3D dans le repère monde.
P_c	Point 3D dans le repère caméra.
$\bar{P}$	Point 3D en coordonnée homogène.
p	Point 2D ou point d'intérêt.
$\bar{p}$	Point 2D en coordonnée homogène.
N	Nombre de poses caméra.
M	Nombre de points 3D.
O	Nombre de points d'intérêt.
C_i	Pose d'indice i, $i \in [1...N]$
P_j	Point 3D d'indice j, $j \in [1...M]$
p_k	Point d'intérêt d'indice k, $k \in [1...O]$
C_{p_i}	Pose observant le point d'intérêt p_i.
P_{p_i}	Point 3D associé au point d'intérêt p_i.
π	La fonction de projection d'un point 3D dans une image.
$0_{3\times1}$	Une matrice de zéros de dimension 3×1.
I_3	Une matrice identité de dimension 3×3.

Introduction

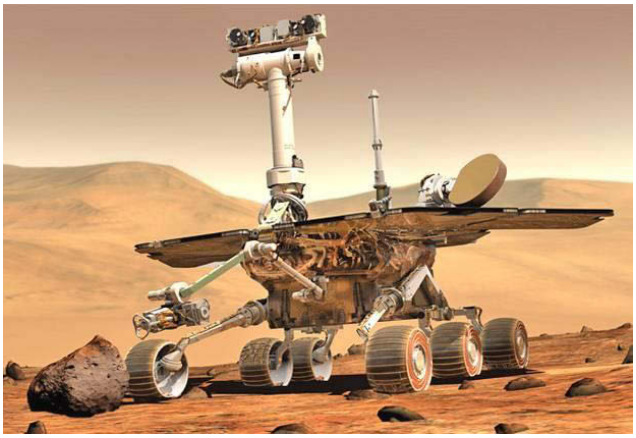
Le SLAM (*Simultaneous Localization And Mapping*) est une technique de SfM (*Structure from Motion*) consistant à localiser un système de perception (*e.g.* une caméra) par rapport à une carte [2] de l'environnement continuellement mise à jour. Cette approche permet l'exploration d'environnements inconnus. La carte est régulièrement actualisée pour intégrer les nouveaux éléments perçus, et la pose du système est calculée en comparant les données observées avec les données déjà présentes dans la carte. Ainsi, les processus de localisation et de mise à jour de la carte sont effectués simultanément de manière dépendante : la localisation utilise le résultat de la cartographie qui a nécessité la connaissance de la pose courante pour géo-référencer les nouvelles données dans la carte existante.

Le SLAM a fait l'objet de nombreuses recherches ces dernières décennies afin d'améliorer la robustesse des méthodes tout en diminuant la puissance de calcul requise. Entre 2004 et 2010, deux véhicules MER (Mars Exploration Rover) ont parcouru 50 km sur Mars. Un algorithme de SLAM visuel, avec 3 paires de caméras et un processeur cadencé à 20 MHz, a permis d'effectuer 10% de cette distance en totale autonomie. D'autres solutions performantes passent le cap de l'industrialisation (certaines sont illustrées sur la figure 1) : aspirateurs automatiques, véhicules sans chauffeur, drones, aides aux personnes malvoyantes, Réalité Augmentée ...

Le SLAM repose sur un système de capteurs et un programme informatique qui traite les données des capteurs. Ainsi, l'amélioration des méthodes de SLAM nécessite soit l'amélioration du système de perception, soit une modélisation plus efficace du problème. Alors que la première solution provoque une augmentation du coût de la solution, la seconde passe par une spécialisation du modèle théorique (géométrique et temporel). Par exemple, il est communément admis que le SLAM s'effectue en environnement statique. Cette hypothèse permet de simplifier le problème, mais d'autres *a priori* peuvent être utilisés. Les MER observent en continu la position du soleil pour limiter la dérive lors de la navigation autonome tel que l'on pourrait le faire avec un magnétomètre ou la gravitation donnée par un accéléromètre. Ces connaissances agissent comme des contraintes sur l'estimation de la trajectoire. Il est également possible d'utiliser des *a priori* sur l'environnement observé : si l'on connaît la pose et la forme d'un objet présent dans la scène, il est avantageux de contraindre le modèle de l'environnement à épouser la forme de l'objet.

De nombreuses contributions ont été proposées dans l'état de l'art pour améliorer le SLAM autant sur l'estimation de la trajectoire que sur la cartographie de l'environnement. Toutefois, aucune solution ne permet à ce jour d'intégrer simplement un ensemble de contraintes hétérogènes dans un même algorithme. Ceci est dû à deux difficultés : les modélisations classiques de trajectoire ne permettent pas d'intégrer des données hétérogènes potentiellement asynchrones sans effectuer d'interpolations ou d'intégrations. De plus, la complexité algorithmique ainsi que la complexité d'implémentation des méthodes d'optimisation constituent un frein à l'utilisation de modélisations trop complexes.

2. Modèle 3D constitué d'amers.



(a) Mars Rover. Crédit : NASA

(b) Aspirateur autonome. Crédit : Samsung

(c) Véhicule autonome. Crédit : Institut Pascal

(d) Réalité Augmentée

(e) Réalité Augmentée sur le château de Léotoing. Crédit : Reoviz

FIGURE 1 – Quelques exemples d'applications de SLAM

Problématique

L'objectif de ce travail est de reconstruire la trajectoire d'un système multi-capteurs en temps réel. Chaque capteur fournit une information sur la trajectoire à une certaine fréquence. Ces informations sont le plus souvent asynchrones et peuvent être de différentes natures : image, position, vitesse, accélération, données de profondeur... Les méthodes de SLAM ont pour objectif d'analyser ces données afin d'estimer la trajectoire du système. Une approche classique pour modéliser la trajectoire consiste à estimer les poses du système aux temps où les données capteurs sont acquises. Toutefois, le temps de traitement des données capteurs est potentiellement important (notamment pour les caméras). C'est pourquoi, il est nécessaire de ne considérer qu'un sous ensemble de ces données et donc un sous ensemble de poses afin d'assurer des performances temps réel, bien que cela puisse dégrader la qualité de la reconstruction. La trajectoire est alors modélisée par un ensemble de poses de référence dont l'espacement temporel est un compromis entre le temps de traitement et la précision du résultat. De plus, les données sont potentiellement asynchrones et de natures différentes (pose, vitesse, accélération,...). Ainsi, l'autre inconvénient de cette modélisation est qu'elle ne permet pas de fusionner les données brutes des capteurs à moins d'avoir recours à des interpolations ou des intégrations.

D'autres informations peuvent être utilisées pour estimer la trajectoire. Il s'agit des *a priori* liés à l'application. Si le système de capteurs se déplace sur un plan, il est avantageux de supprimer un degré de liberté à la trajectoire ; ou lorsque le système est fixé sur un véhicule, un modèle d'évolution approprié facilitera l'estimation de la trajectoire. En supposant l'environnement rigide et constitué d'objets partiellement connus, des méthodes de reconstruction 3D produiront un modèle de l'environnement utilisable pour une meilleure localisation du système. Ainsi, les connaissances *a priori* apporte une meilleure interprétation des données capteurs et apportent des contraintes supplémentaires au processus d'estimation de la trajectoire. Toutefois, le nombre de contraintes potentiellement important rend difficile l'implémentation du processus d'optimisation de la trajectoire.

C'est pourquoi, l'objectif du travail présenté ici est de proposer à la fois un modèle de SLAM à contraintes multiples intégrant des données hétérogènes et une solution performante pour la résolution du processus d'optimisation associé. En particulier, la solution proposée doit fusionner facilement, dans un algorithme de SLAM, des contraintes provenant de différents capteurs (caméra, centrale inertielle, GPS ...) ou d'*a priori* liés à l'application (modèle d'évolution ou présence d'objets 3D dans la scène) tout en conservant des performances d'exécution temps réel. Les deux problèmes majeurs que l'on identifie ici sont : 1) comment fusionner en temps réel des données hétérogènes potentiellement asynchrones dans un processus d'optimisation pour estimer une même trajectoire, et 2) comment générer un processus d'optimisation performant et capable d'intégrer simplement toutes les contraintes du problème ? De plus, la solution proposée doit garantir une faible consommation en ressource matérielle tout en étant facile à déployer donc portable sur les architectures les plus communes (ordinateur personnel, tablette, *smartphone* ...). C'est pourquoi, nous limitons au maximum l'utilisation explicite d'accélération matérielle (*ie.* SIMD, programmation parallèle, GPU, grille de calcul ...).

Contributions

L'algorithme de SLAM à contraintes multiples présenté dans ce manuscrit a permis la réalisation de deux contributions majeures. La première est un algorithme SLAM à contraintes multiples nommé MCSLAM. Les contraintes sont utilisées afin d'améliorer la précision et la robustesse du SLAM. Elles interviennent à la fois sur le mouvement du système et le modèle d'environnement. Elles proviennent soit du système multi-capteurs, soit d'*a priori* comme des objets 3D partiellement connus. L'estimation du mouvement, modélisé par une trajectoire continue, se fait en temps réel en intégrant l'ensemble des contraintes dans un même processus d'optimisation. La seconde contribution est une implémentation performante et simple d'utilisation de l'algorithme de Levenberg-Marquardt pour l'optimisation de problèmes à contraintes multiples. Ces contributions ont donné lieu aux publications ci-dessous.

Publications dans le cadre de cette thèse :

[6] Datta Ramadasan, Clément Deymier, Marc Chevaldonné, and Thierry Chateau. SLAM visuel temps réel pour l'estimation précise de plan. In *Reconnaissance de Formes et Intelligence Artificielle*, RFIA 2014.

[7] Datta Ramadasan, Marc Chevaldonné, and Thierry Chateau. SLAM contraint en environnement de grande taille. In *Reconnaissance de Formes et Intelligence Artificielle*, RFIA 2014.

[8] Datta Ramadasan, Marc Chevaldonné, and Thierry Chateau. Real-time SLAM for static multi-objects learning and tracking applied to Augmented Reality applications. In *Virtual Reality*, VR 2015.

[9] Datta Ramadasan, Marc Chevaldonné, and Thierry Chateau. DCSLAM : un SLAM temps réel à contraintes dynamiques. In *Journées francophones des jeunes chercheurs en vision par ordinateur*, ORASIS 2015.

[10] Datta Ramadasan, Marc Chevaldonné, and Thierry Chateau. DCSLAM : A Dynamically Constrained real-time SLAM. In *International Conference of Image Processing*, ICIP 2015.

[11] Datta Ramadasan, Marc Chevaldonné and Thierry Chateau. MCSLAM : a Multiple Constrained SLAM. In *Proceedings of the British Machine Vision Conference (BMVC)*, pages 107.1-107.12. BMVA Press, September 2015.

Publications antérieures à la thèse :

[12] François Bardet, Thierry Chateau, Datta Ramadasan. Suivi et catégorisation visuels en temps réel d'un nombre variable d'objets : application au suivi de véhicules. In *Congrès des jeunes chercheurs en vision par ordinateur*, ORASIS 2009.

[13] François Bardet, Thierry Chateau, and Datta Ramadasan. Real-time multi-object tracking with few particles. In *International Conference on Computer Vision Theory and Applications*, pages 456–463, VISAPP 2009.

[14] François Bardet, Thierry Chateau, and Datta Ramadasan. Unifying real-time multi-vehicle tracking and categorization. In *Intelligent Vehicles Symposium*, pages 197–202. IV 2009.

[15] François Bardet, Thierry Chateau, and Datta Ramadasan. Illumination aware MCMC particle filter for long-term outdoor multi-object simultaneous tracking and classification. In *IEEE International Conference on Computer Vision*, pages 1623–1630, ICCV 2009.

[16] Nadir Karam, Hicham Hadj-Abdelkader, Clement Deymier, and Datta Ramadasan. Improved visual localization and navigation using proprioceptive sensors. In *IEEE/RSJ Intelligent Robots and Systems*, pages 4155–4160. IROS 2010.

[17] Brevet 2012) : Method of calibrating a computer-based vision system onboard a craft.

(a) Feder (b) Europe

FIGURE 2 – Organismes à l'origine de la bourse innovation

Contexte

Les travaux de cette thèse ont été effectués dans le cadre d'une bourse innovation financée par la Région Auvergne et le Fonds Européen de Développement Régional dont les logos sont présentés par les figures 2a et 2b. La thèse a été menée au sein de l'équipe de vision par ordinateur *ComSee* de l'axe thématique ISPR (Image, Systèmes de Perception, Robotique) de l'Institut Pascal. L'équipe travaille principalement sur des problématiques de localisation et reconstruction 3D, de calibrage de capteurs et de suivi visuel. Le sujet initial de la thèse concernait les méthodes de recalage d'objets 3D en temps réel sur un flux vidéo pour des applications de Réalité Augmentée sur plate-formes mobiles. C'est pourquoi, une attention particulière a été apportée pour que les méthodes développées soient performantes et utilisables sur des architectures limitées en ressources. La problématique a évolué au cours de la thèse pour traiter le cas plus général du SLAM à contraintes multiples. La problématique originale est alors un sous problème du nouveau sujet. Pour des raisons pratiques, la majorité des développements et expérimentations a été réalisée sur un ordinateur de bureau [3] avec une caméra *global shutter* et des contraintes de reconstruction simples. Les expérimentations illustrent différents algorithmes de SLAM contraints réalisés sur la base d'un SLAM visuel temps réel. Celui-ci nécessite une caméra vidéo *global shutter* calibrée dans un environnement rigide partiellement connu et analyse en continu le flux vidéo. Les contraintes additionnelles proviennent de différents capteurs (centrale inertielle ou GPS) et/ou d'*a priori* sur la structure 3D de l'environnement (présence d'objets 3D partiellement connus dans la scène). Toutefois, l'approche se veut portable et générique sur les types de caméras et de contraintes.

Structure du document

Ce document commence par un rappel des notions nécessaires à la compréhension des méthodes présentées dans les parties suivantes. Puis, une première partie présente un état de l'art du SLAM par vision monoculaire ainsi que les différentes étapes de l'algorithme de SLAM visuel réalisé au cours de cette thèse. Cet algorithme sert de base aux expérimentations sur le SLAM contraint. La seconde partie présente les bibliothèques d'optimisation existantes ainsi que la difficulté liée à l'implémentation efficace de tels outils. Puis le fonctionnement et les caractéristiques techniques de la bibliothèque LMA sont présentés. Cette partie conclut sur des comparatifs entre LMA et quelques bibliothèques de l'état de l'art sur divers problèmes d'optimisation. La troisième partie concerne l'algorithme de SLAM à contraintes multiples nommé MCSLAM. Les concepts de contraintes hétérogènes et asynchrones sur une trajectoire continue ainsi que les contraintes sur la structure 3D de la reconstruction sont alors détaillés. Des résultats qualitatifs et quantitatifs évaluent l'utilisation du MCSLAM dans le cadre d'applications réelles de Réalité Augmentée et de navigation autonome.

3. architecture multi-cœurs x86 supportant les instructions vectorielles et dépourvu de GPU

Chapitre 1

Notions de base

Ce chapitre décrit les notions de base couramment utilisées dans le domaine de la reconstruction 3D par vision et nécessaires à la compréhension de la suite du manuscrit. Une première section présente la notion de changement de repère nécessaire à la compréhension des modèles de caméra présentés ensuite. Une autre section décrit le calcul de l'erreur de reprojection utilisant les modèles de caméras. Puis, les méthodes d'optimisations continues sont présentées. Ces méthodes sont utilisées pour estimer les paramètres intrinsèques et extrinsèques d'une caméra en minimisant l'erreur de reprojection. La section qui suit présente une méthode de détection de points d'intérêt ainsi qu'une méthode de mise en correspondance basée sur l'apparence des points. Des outils géométriques utilisant les mises en correspondance entre images pour obtenir des informations relatives à la pose des amers visuels dans l'espace 3D sont ensuite présentés. Le chapitre se termine sur une présentation des notions de programmation nécessaires à la compréhension de la partie II.

1.1 Changement de base et matrice de passage

Le changement de repère se compose d'une rotation et d'une translation. Toutefois, il existe plusieurs conventions. Il est possible d'appliquer d'abord la rotation, puis la translation ou inversement. De même, selon les sources, les rotations s'appliquent soit au déplacement du repère, soit au déplacement du vecteur translation. C'est pourquoi nous définissons ici la convention utilisée pour l'ensemble du manuscrit.

Soit $\mathcal{B}_1$, $\mathcal{B}_2$ et $\mathcal{B}_3$ trois bases de $\mathbb{R}^n$ et X_1, X_2, X_3 les représentations d'un vecteur x respectivement dans ces différentes bases en coordonnées homogènes.

Changement de base On appelle matrice de passage homogène de la base $\mathcal{B}_1$ à la base $\mathcal{B}_2$, la matrice T_1^2 vérifiant :

$$X_2 = T_1^2 X_1 \tag{1.1}$$

Cette formule permet d'obtenir le vecteur exprimé dans $\mathcal{B}_2$ à partir de celui exprimé dans $\mathcal{B}_1$.

Composition de changements de base Soit T_1^3 la matrice de passage de la base $\mathcal{B}_1$ à $\mathcal{B}_3$ et T_2^3 la matrice de passage de la base $\mathcal{B}_2$ à $\mathcal{B}_3$. Leur composition s'exprime trivialement par :

$$T_1^3 = T_1^2 T_2^3 \tag{1.2}$$

FIGURE 1.1 – Illustration de la distorsion sur une image 640×480 acquise avec un objectif grand angle (champ de vue de 182° et focale de 1.4 mm).

Les transformations euclidiennes Soit R et t respectivement une matrice de rotation et un vecteur de translation entre les bases $\mathcal{B}_1$ et $\mathcal{B}_2$. La transformation euclidienne entre ces bases s'exprime :

$$T_1^2 = \left[\begin{array}{c|c} R & t \\ \hline 0_{3\times1} & 1 \end{array}\right] \tag{1.3}$$

1.2 Modèles de caméra

Il existe différents types de caméras. En vision par ordinateur, la plupart dispose au moins d'un système optique (objectif) et d'un capteur photographique (CCD, CMOS). La lumière traverse les lentilles et passe dans l'iris. L'image se forme sur le capteur où elle est enregistrée. Les objectifs montés devant le capteur permettent une focalisation de la lumière. Toutefois, cela peut engendrer une distorsion de l'image comme le montre la figure 1.1. Ainsi, les droites deviennent courbes et semblent incurvées vers l'intérieur, on parle de distorsion en coussinet. Si elles sont incurvées vers l'extérieur on parle de distorsion en barillet. Dans le cadre de reconstruction 3D, la distorsion géométrique doit être modélisée ou corrigée.

Les modèles intrinsèques de caméra approximent le processus physique de formation de l'image à partir de l'observation de l'environnement. Ils associent un objet de la scène à un pixel de l'image en modélisant l'étape de projection. Nous présentons ici deux modèles couramment utilisés et permettant de prendre en compte la distorsion : le modèle sténopé avec correction de la distorsion et le modèle unifié. Pour plus d'informations sur les modèles de caméra, le lecteur intéressé peut se rapporter à la description proposée par Sturm *et al.* dans [18].

1.2.1 Modèle sténopé

L'une des modélisations les plus communes est le modèle sténopé ou *pinhole* qui est un modèle central. Il possède un centre optique unique. Le point image est défini par l'intersection de la droite

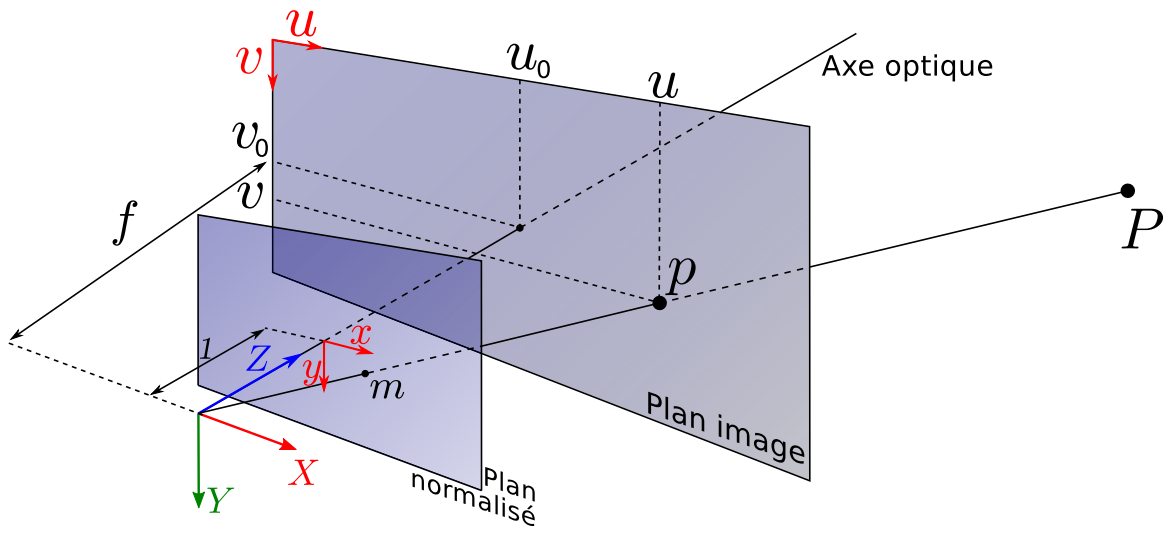


FIGURE 1.2 – Illustration du fonctionnement du modèle de projection sténopé.

reliant un point 3D et le centre optique avec un plan situé à une distance focale f. Cette modélisation porte le nom de *pinhole* car elle se comporte exactement comme la lumière qui traverserait une feuille trouée par une épingle. La lumière entre par le trou et forme l'image sur un écran disposé derrière selon une trajectoire rectiligne.

Par convention, les axes du repère de la caméra sont choisis tels que X soit orienté vers la droite, Y vers le bas et Z vers l'avant, dans la même direction que l'axe optique. L'image se focalise à une distance $Z = -f$. Par simplicité, on utilise une symétrie centrale pour placer le plan image tel que $Z = f$. On définit le repère pixellique (u, v) dont l'origine est située dans le coin supérieur gauche de l'image, u vers la droite et v vers le bas. On appelle plan normalisé le plan tel que $Z = 1$. Les points 2D appartenant à ce plan sont exprimés dans le repère (x, y) *cf.* figure 1.2.

Projection

La projection d'un point 3D P_w (exprimé dans le repère monde) dans le plan image en un point p s'effectue en trois étapes successives, illustrées par la figure 1.2 (sur laquelle les points P_w et P_c sont représentés par le point P) :

1. Le point $\bar{P_w} = (X_w, Y_w, Z_w, 1)$ en coordonnées homogènes est placé dans le repère de la caméra afin d'obtenir $\bar{P_c}$ grâce à un changement de base expliqué dans la section 1.1.
2. Le point $\bar{P_c}$ est exprimé dans le plan normalisé de la caméra, ce qui s'écrit en coordonnées homogènes :

$$\bar{m} = \left[\; I_3 \; \middle| \; 0_{3\times 1} \; \right] \bar{P_c} \tag{1.4}$$

3. Pour passer du repère caméra au repère image (coordonnées en pixels), on utilise la transformation affine suivante :

$$\bar{p} = K\bar{m} \tag{1.5}$$

avec :

$$K = \begin{pmatrix} f_x & 0 & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{pmatrix} \tag{1.6}$$

qui représente la matrice des paramètres intrinsèques constituée de f_x et f_y les focales verticale et horizontale de la caméra, u_0 et v_0 les coordonnées cartésiennes de la position du centre optique dans l'image.

1.2.2 Modèle sténopé avec distorsion radiale polynomiale

Les distorsions géométriques déforment les images. Elles sont composées d'une distorsion radiale et une tangentielle. La composante tangentielle est considérée négligeable car infime. La composante radiale peut être modélisée par un polynôme [18]. Il existe deux conventions dans la littérature :

1. Le polynôme permettant de passer des coordonnées non distordues m_u aux coordonnées distordues m_d. Ce modèle sera appelé *modèle direct*. Il est utilisé dans la bibliothèque OpenCV [1].
2. Le polynôme permettant de passer des coordonnées distordues aux coordonnées non distordues. Ce modèle qualifié standard dans la littérature est appelé *modèle indirect*.

On modélise la déformation de l'objectif en utilisant un modèle de distorsion radiale exploitant un polynôme I de degré 10 afin de représenter avec précision la fonction. La déformation étant symétrique, seuls les coefficients pairs ont une valeur non-nulle que l'on note $(a_1, a_2, a_3, a_4, a_5)$.

Modèle direct

En utilisant le modèle direct, la projection d'un point 3D P en un point distordu p_d dans le plan image est obtenue de manière "directe" en trois étapes (illustrées par la figure 1.3) :

1. Projection perspective du point 3D P en m_u (*cf.* équation 1.4),
2. Application de la distorsion sur m_u par transformation polynomiale pour obtenir m_d :

$$m_d = (1 + D(r_u))m_u \tag{1.7}$$

avec le polynôme

$$D(r_u) = \sum_{k=1}^{5} a_k r_u^{2k} \tag{1.8}$$

et la distance radiale entre le centre optique (u_0, v_0) et le point $m_u = (m_{u_x}, m_{u_y})$ dans le plan normalisé

$$r_u = \sqrt{(m_{u_x} - u_0)^2 + (m_{u_y} - v_0)^2} \tag{1.9}$$

3. Application des paramètres intrinsèques K sur m_d par transformation affine pour obtenir p_d (*cf.* équations 1.5 et 1.6).

Modèle indirect

Le polynôme I permet de connaître la modification de la distance entre le point principal et la projection d'un point sur le plan image afin de le distordre. La figure 1.3 illustre l'application de cette correction à un point image noté m_d. Concrètement, pour obtenir un point sans distorsion m_u à partir du point distordu m_d, on suit les étapes suivantes :

1. On calcule $r_d = \sqrt{x_d^2 + y_d^2}$ la distance radiale entre l'axe optique et le point m_d dans le plan normalisé.
2. On estime le polynôme radial pour la valeur correspondante :

$$I(r_d) = \sum_{k=1}^{5} a_k r_d^{2k} \tag{1.10}$$

1. bibliothèque libre pour la vision par ordinateur : opencv.org

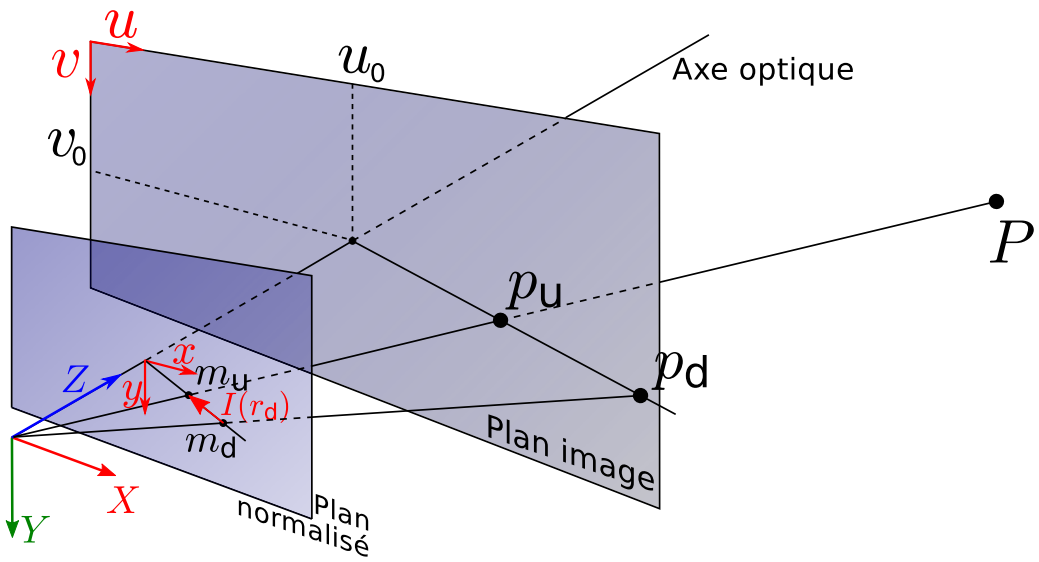


FIGURE 1.3 – Illustration du modèle de projection sténopé avec distorsion radiale.

3. On incrémente m_d afin d'obtenir m_u grâce à la relation :

$$m_u = (1 + I(r_d))m_d \tag{1.11}$$

Discussion

La distorsion polynomiale radiale possède l'avantage d'être un modèle fiable et permettant de corriger les amers visuels juste après leur détection. La normalisation par la focale permet de le rendre invariant lorsque les images sont redimensionnées ou rognées.

Cependant, il souffre de défauts majeurs : premièrement, le nombre de paramètres est important au regard du phénomène de distorsion lui-même. En effet, on constate qu'en faisant varier les valeurs des paramètres, il est aisé de construire des distorsions physiquement exubérantes. De plus, le processus inverse consistant à retrouver le rayon sur lequel le point 3D devrait se trouver à partir d'un point de l'image n'admet pas de solution analytique car un polynôme de degré supérieur à 5 ne possède pas de racine analytique. Cette procédure nécessite donc l'utilisation d'un algorithme itératif de minimisation et de recherche dichotomique.

Enfin, lorsque le modèle indirect est utilisé pour distordre des primitives détectées sur une image, il entraîne un biais. De nombreux algorithmes de vision par ordinateur supposent que les amers visuels détectés ont un bruit gaussien et centré afin d'exprimer les problématiques en terme de vraisemblance probabiliste. Cependant, lors de la correction de la distorsion, les densités de probabilité sont déformées et ne correspondent plus au bruit de mesure original. Ceci limite son usage aux cas de faible distorsion.

1.2.3 Modèle unifié

Initialement construit pour les caméras catadioptriques, le modèle unifié [19] peut aussi bien modéliser, avec le même formalisme mathématique, les caméras perspective que les caméras grand-angle de type *Fish-Eye*. Contrairement au modèle sténopé dont l'étape de projection s'effectue directement sur le plan focal, le modèle unifié passe par une projection sur la sphère puis une seconde phase de projection sur un plan. Cette solution originale est une bonne modélisation des phénomènes physiques donnant lieu à la distorsion géométrique. De ce fait, on remarque que ce modèle possède un seul et

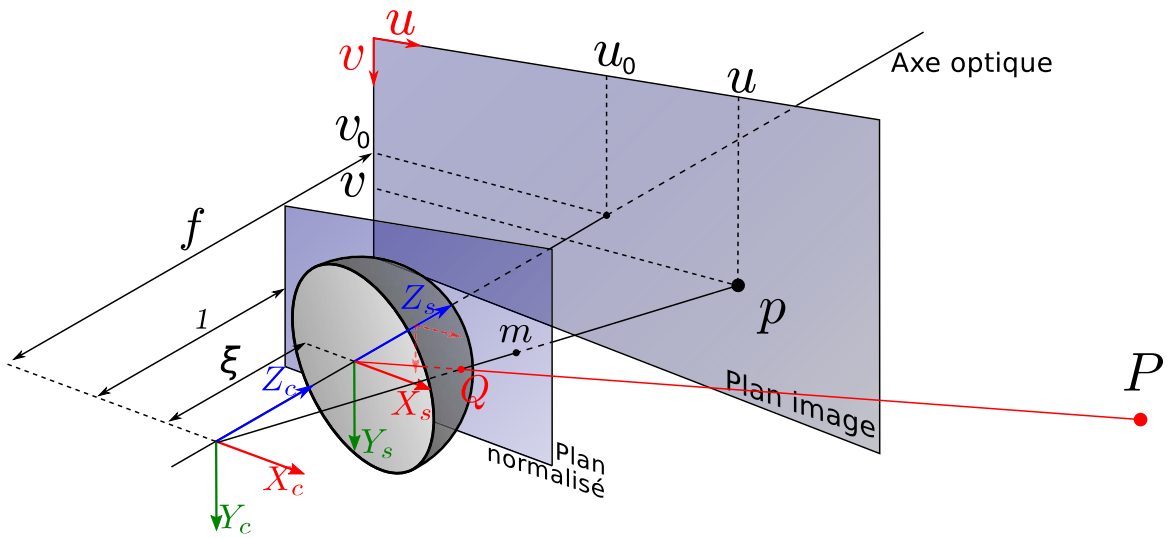


FIGURE 1.4 – Illustration des notations employées et du principe du modèle unifié.

unique paramètre ξ pour la distorsion tandis que 5 sont nécessaires avec les modèles sténopés à distorsion polynomiale. Une illustration des notations employées et de son fonctionnement est disponible à la figure 1.4.

Projection

On note P_w un point 3D dont les coordonnées sont exprimées dans le repère monde. Quatre étapes sont nécessaires à sa projection :

1. Le point 3D $P_w = (X_w, Y_w, Z_w, 1)$ exprimé dans le repère de la caméra est noté P_c. Le changement de repère est presenté dans la section 1.1.
2. Le point P_c est projeté sur la sphère unitaire de centre $Zc = \xi \leq 0$. Pour cela, on note $P_s = (X_c, Y_c, Z_c - \xi)$ les coordonnées de P_c dans le repère de la sphère, le point normalisé Q s'exprime alors :

$$Q = \frac{P_s}{\rho} = \frac{1}{\rho}\begin{pmatrix} X_s \\ Y_s \\ Z_s \end{pmatrix} \tag{1.12}$$

   avec $\rho = \sqrt{X_s^2 + Y_s^2 + Z_s^2}$
3. Puis la projection de Q sur le plan normalisé $Z_c = 1$ en un point $\bar{m}_c$ par :

$$\bar{m}_c = \begin{pmatrix} \frac{X_s}{Z_s+\rho\xi} \\ \frac{Y_s}{Z_s+\rho\xi} \\ 1 \end{pmatrix} \sim \begin{pmatrix} \frac{X_s}{\rho} \\ \frac{Y_s}{\rho} \\ \frac{Z_s}{\rho} + \xi \end{pmatrix} \tag{1.13}$$

4. Enfin la transformation affine pour obtenir le point projeté p.

$$\bar{p}_c = K\bar{m}_c \tag{1.14}$$

   Avec $K = \begin{pmatrix} f_x & 0 & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{pmatrix}$ la matrice des paramètres intrinsèques

Lancer de rayon

L'opération de lancer de rayon (ou rétroprojection) consiste, à partir d'un point de l'image, à retrouver le rayon 3D sur lequel se trouve le point de l'espace correspondant. Cette opération inverse de la projection est utilisée notamment pour la triangulation de primitives. Le modèle unifié permet d'effectuer cette opération facilement, car il est "inversible" au sens où il est possible de retrouver analytiquement le point sur la sphère, donc le rayon QP à partir d'un point de l'image p par la succession d'opérations suivantes :

1. On retrouve le point m sur le plan image normalisé à partir du point image p grâce à la relation :

$$\bar{m}_c = \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = K^{-1}\bar{p}_c \tag{1.15}$$

2. Puis le point Q_s sur la sphère par :

$$Q_s = \nu \begin{pmatrix} x \\ y \\ 1 - \nu^{-1}\xi \end{pmatrix} \tag{1.16}$$

Avec

$$\nu = \frac{\xi + \sqrt{1 + (x^2 + y^2)(1 - \xi^2)}}{x^2 + y^2 + 1} \tag{1.17}$$

3. Enfin une transformation 3D est appliquée afin de restituer le rayon dans le repère monde.

1.2.4 Conclusion

L'utilisation du modèle sténopé avec distorsion radiale pose des difficultés (nombre de paramètres importants, processus itératif de correction et biais sur la précision des détections) pour la modélisation de caméra grand angle au sein de processus de reconstruction 3D. C'est pourquoi, le modèle unifié est celui qui sera utilisé dans le reste de nos travaux.

1.3 Erreur de reprojection

L'erreur de reprojection ϵ correspond à la distance euclidienne entre le projeté d'un point 3D P_w dans l'image et l'observation p de ce même point obtenue par détection (*cf.* figure 1.5). Le modèle de projection π (*cf.* section 1.2) utilise la pose caméra $\{R, t\}$ exprimée dans le repère monde et associée à l'image. Lorsque les données du problème, *ie.* l'ensemble des paramètres $X = \{R, t, P_w\}$ ainsi que le modèle de projection π et le point d'intérêt p, sont précisément connus, l'erreur de reprojection ϵ est nulle. Ainsi le modèle et les paramètres concordent précisément avec la donnée observée :

$$\pi(X) = p \tag{1.18}$$

En pratique, c'est rarement le cas, à cause des erreurs d'estimation des paramètres X et du bruit de détection sur p, on peut alors mesurer l'erreur de reprojection ϵ :

$$\pi(X) = p - \epsilon \tag{1.19}$$

Le bruit de détection étant considéré gaussien, on peut utiliser un processus d'optimisation des paramètres X qui minimise le carré des erreurs de reprojection, on parle alors d'ajustement de faisceaux :

$$\hat{X} = \underset{X \in \mathbb{R}^{\mathbb{N}}}{\arg\min} \parallel p - \pi(X) \parallel^2 \tag{1.20}$$

Les méthodes d'optimisation sont abordées dans la section 1.5.

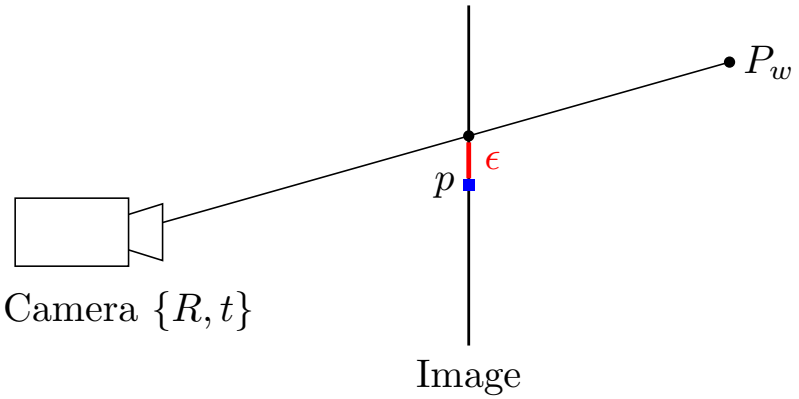


FIGURE 1.5 – Illustration de l'erreur de reprojection ϵ entre une détection p et la projection d'un point 3D P_w dans l'image acquise par une caméra dont la pose est définie par une rotation R et une translation t.

1.4 Représentation exponentielle locale

Les rotations de l'espace 3D sont les éléments du groupe special orthogonal, appelé $\mathbb{SO}3$ et assimilable aux matrices réelles orthogonales de taille 3. Cependant, cette représentation du groupe possède neuf valeurs alors que $\mathbb{SO}3$ est intrinsèquement de dimension 3. L'optimisation des 9 coefficients d'une matrice de rotation n'est pas envisageable en raison des non linéarités entre les coefficients et des nombreuses contraintes corrélant leurs évolutions. C'est pourquoi, il convient d'utiliser une paramétrisation fournissant une représentation minimale, isotrope et pratique à utiliser.

Nous choisissons d'utiliser la paramétrisation de carte exponentielle locale développée par [20] et [21] dans le cadre d'optimisation sur les variétés. La carte exponentielle consiste en une application permettant de passer de la variété à un espace dit « tangent » qui se comporte comme un espace affine au voisinage d'un point M_1 (voir la figure 1.6). De plus, la carte exponentielle de la variété conserve les distances et les géodésiques dont l'origine est le point M_1 de la variété. Identiquement au cas de la translation, on utilise un espace (le plan tangent) où l'ajout d'un incrément modifie M_1 selon un tracé géodésique. Une fois l'incrément appliqué, il est nécessaire de redéfinir un nouveau plan tangent qui est localement exploitable.

$\mathbb{SO}3$ étant un groupe de Lie, le plan tangent est une algèbre de Lie $\mathfrak{so}(3)$ représentée par l'ensemble des matrices antisymétriques de taille 3 que nous paramétrerons par :

$$\omega = \begin{pmatrix} 0 & -w_z & w_y \\ w_z & 0 & -w_x \\ -w_y & w_x & 0 \end{pmatrix} \tag{1.21}$$

si l'espace tangent est défini pour un élément M_1 de la variété. La matrice ω représente un déplacement selon une géodésique (voir figure 1.6) de la variété jusqu'à un nouvel élément M_2 dont l'expression est :

$$M_2 = M_1 \exp(\omega) \tag{1.22}$$

Avec :

$$\exp(\omega) = \sum_{k=0}^{\infty} \frac{(\omega)^k}{k!} = I + \omega + \frac{1}{2!}(\omega)^2 + \frac{1}{3!}(\omega)^3 + \cdots \tag{1.23}$$

Une fois la nouvelle matrice M_2 obtenue, w_x,w_y,w_z reprennent la valeur 0 car ils définissent maintenant un nouveau plan tangent en M_2. Cette représentation locale permet de décrire les rotations avec seulement trois paramètres, elle ne souffre pas du problème de blocage de cardan ni d'aucune singularité

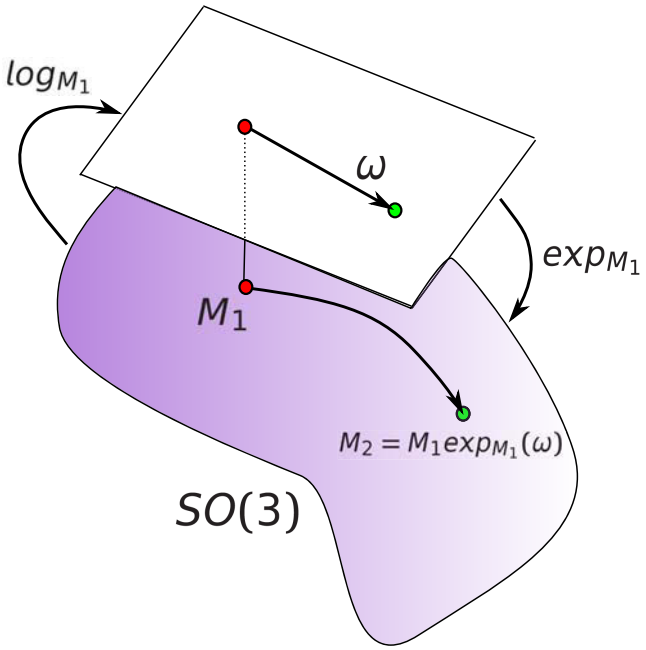


FIGURE 1.6 – La carte exponentielle locale permet de passer de l'espace tangent en blanc à la variété $\mathbb{SO}3$ en mauve à l'aide de l'exponentielle matricielle. L'espace tangent permet de travailler dans un espace affine où les opérations usuelles sont disponibles tandis que l'exponentielle est utilisée pour retourner sur la variété en garantissant qu'un trajet rectiligne dans l'espace tangent est effectué sur la variété selon une géodésique.

de paramétrisation. L'estimation de l'exponentielle matricielle peut s'effectuer plus rapidement qu'en utilisant la série entière grâce à la formule de Rodrigues :

$$\exp(\omega) = I + \omega \sin(\theta) + \omega^2(1 - \cos(\theta)) \tag{1.24}$$

Où θ est l'angle de rotation défini par la norme de la matrice w. Pour les très petits angles on peut utiliser l'approximation au premier ordre de l'exponentielle donnée par :

$$M_2 = M_1(I + \omega) \tag{1.25}$$

Inversement, si on veut exprimer une matrice M_2 dans le plan tangent local à M_1, il faut recaler M_2 dans le repère de M_1 puis utiliser le logarithme matriciel :

$$\omega_1 = \log(M_1^T M_2) \tag{1.26}$$

qui peut être lui aussi estimé par une formule de Rodrigues :

$$\log(R) = \begin{cases} 0 & \text{si } \theta = 0 \\ \frac{\theta}{2\sin(\theta)}(R - R^\top) & \text{si } \theta \neq 0 \text{ et } \theta \in (-\pi, \pi) \end{cases} \tag{1.27}$$

Cette paramétrisation offre une représentation minimale, évite le blocage de cardan et est numériquement stable. C'est pourquoi, nous utilisons dans l'ensemble de nos travaux la représentation exponentielle locale pour paramétrer les rotations 3D.

1.5 Optimisation

Les méthodes d'optimisation consistent à trouver, parmi l'ensemble des solutions d'un problème, celle qui satisfait au mieux un critère donné. Le plus souvent, on recherche le minimum d'une fonction, donc la solution correspondant à l'ensemble de paramètres telle que la fonction soit minimale en ce point.

1.5.1 Moindres carrés linéaires

La méthode des moindres carrés permet d'estimer l'ensemble des paramètres d'un modèle mathématique à partir d'une liste d'observations entachées d'erreurs de mesure. Étant donné F un modèle mathématique, et Y un ensemble d'observations, il s'agit d'estimer l'ensemble de paramètres $\hat{X}$ minimisant l'erreur entre les données estimées par le modèle et les observations Y :

$$\hat{X} = \arg\min_{X} \sum_{i=0}^{n-1} \|Y_i - F(X_i)\|^2 \tag{1.28}$$

1.5.2 Moindres carrés non linéaires

Les méthodes de moindres carrés non linéaires consistent à trouver le minimum d'une fonction non-linéaire de manière itérative en supposant le problème localement linéaire. Ainsi, chaque itération se compose d'une recherche des directions de descente de gradient, autrement dit la jacobienne J de la fonction, et d'un pas à appliquer aux paramètres X pour minimiser le vecteur d'erreur ϵ de la fonction. La matrice jacobienne est la matrice des dérivées partielles du premier ordre d'une fonction vectorielle. Autrement dit, l'élément d'indices (i, j) de la jacobienne correspond à la dérivée partielle de la i$^{\text{ème}}$ composante par rapport à la j$^{\text{ème}}$ variable. Etant donné une fonction $\mathcal{E}$

$$\mathcal{E}(X) = F_1(X) + ... + F_n(X) \tag{1.29}$$

telle que

$$\mathcal{E}(X) = \sum_{i=0}^{N_1} f_{1,i}(X) + ... + \sum_{i=0}^{N_n} f_{n,i}(X) \tag{1.30}$$

on peut définir la jacobienne J de la fonction $\mathcal{E}$ par :

$$J = \partial\mathcal{E} = \left[\frac{\partial F_1}{\partial X_{F_1}}, \ldots, \frac{\partial F_n}{\partial X_{F_n}}\right] \tag{1.31}$$

avec X_{F_i} l'ensemble des paramètres intervenant dans la fonction F_i. L'utilisation des méthodes de moindres carrés non linéaires implique donc le calcul des dérivées de la fonction.

Algorithme du gradient

L'algorithme du gradient est une méthode d'optimisation de premier ordre à direction de descente. La direction de déplacement choisie à chaque itération dépend du gradient de la fonction à minimiser. Le facteur multiplicateur γ_k, correspondant au pas de l'itération k, peut être obtenu par recherche linéaire [22] (*Line search method*) pour garantir la convergence, mais le pas obtenu n'est pas forcément optimal. A l'itération k, l'algorithme du gradient met à jour le vecteur de paramètres en utilisant les dérivées comme direction de descente :

$$X_{k+1} = X_k - \gamma_k J_k^T \epsilon_k \tag{1.32}$$

Cette méthode a l'avantage de converger même en cas de mauvaise initialisation du système à résoudre. Toutefois, la convergence peut s'avérer extrêmement lente si les directions de descente de la fonction sont variables.

Algorithme de Newton-Raphson

L'algorithme de Newton-Raphson est une méthode itérative de second ordre (donc à convergence rapide) basée sur l'algorithme de Newton. Il s'agit de résoudre à chaque itération k le système d'équations suivant en calculant la hessienne H de la fonction de coût :

$$X_{k+1} = X_k - H^{-1} J_k^T \epsilon_k \tag{1.33}$$

Contrairement à la descente de gradient, cet algorithme est d'autant plus efficace qu'il se rapproche de la bonne solution et l'incrément obtenu est localement optimal. Toutefois, cette méthode nécessite le calcul de la hessienne (donc des dérivées secondes) de la fonction de coût ainsi qu'une initialisation proche de la solution recherchée.

Algorithme de Gauss-Newton

L'algorithme de Gauss-Newton est une approximation de l'algorithme de Newton-Raphson où la matrice des dérivées secondes H est approximée par $J^T J$ épargnant ainsi le calcul des dérivées secondes. Le système à résoudre est alors le suivant :

$$X_{k+1} = X_k - (J_k^T J_k)^{-1} J_k^T \epsilon_k \tag{1.34}$$

Cette méthode possède un avantage certain car il n'est pas toujours possible de calculer les dérivées secondes. Toutefois, la hessienne étant approximée, cette méthode converge moins rapidement que la méthode de Newton-Raphson.

Levenberg-Marquardt

L'algorithme de Levenberg-Marquart [23, 24] tend à combiner les avantages des algorithmes de Gauss-Newton et du gradient en augmentant la diagonale de $J_k^T J_k$ par un paramètre λ_k dont la valeur est ajustée empiriquement à chaque itération. Le système à résoudre à chaque itération k est le suivant :

$$X_{k+1} = X_k - (J_k^T J_k + \lambda_k I)^{-1} J_k^T \epsilon_k \tag{1.35}$$

Le paramètre λ_k est mis à jour à chaque itération en fonction de la variation de l'erreur :

- lorsque le coût de la fonction diminue, on diminue la valeur de λ_k qui tend à se rapprocher de 0, le comportement de la méthode est alors proche de l'algorithme de Gauss-Newton,
- lorsque le coût de la fonction augmente, on augmente la valeur de λ_k rendant prépondérantes les valeurs sur la diagonale de la hessienne et menant l'algorithme à un comportement de type descente de gradient.

L'algorithme de Levenberg-Marquardt est détaillé par l'algorithme 1 avec une méthode de mise à jour de λ proposée par Nielsen [25] et couramment utilisée dans l'état de l'art. L'utilisation d'une méthode descente de gradient ou de Gauss-Newton dépend principalement de la proximité de la solution initiale à la solution optimale. C'est pourquoi, l'algorithme de Levenberg-Marquardt, évoluant au cours des itérations vers la méthode la plus adaptée, est la méthode que nous choisissons d'utiliser.

1.6 Amers visuels

Il existe différents types de primitives : des régions, des contours ou des points. Les primitives de type points sont couramment utilisées avec les opérations géométriques telles que le calcul de pose, la géométrie épipolaire et l'ajustement de faisceaux. De plus, cette primitive ponctuelle possède deux propriétés intéressantes : d'une part, elle fixe tous les degrés de liberté dans l'image contrairement aux

Algorithm 1: Algorithme de Levenberg-Marquardt

Data: Une fonction $\mathcal{E} : \mathbb{R}^M \to \mathbb{R}^N$ avec $N \geq M$, $\mathcal{E}(X) \in \mathcal{R}^N$ le vecteur d'erreurs et $X \in \mathcal{R}^M$ le vecteur de paramètres

Result: Un vecteur de paramètres $\hat{X}$ tel que $\|\mathcal{E}(\hat{X})\|$ soit minimale

$it = 0$;$v = 2$;$\lambda = 0.001$;
$\Delta = \texttt{Zeros}(M)$;
$J = \texttt{Zeros}(N, M)$;
while $it \leq 20$ **do**
  $\epsilon_1 = \mathcal{E}(X)$;
  `Solve` $(J^T J + \lambda I)\Delta = J^T \mathcal{E}(X)$;
  $\epsilon_2 = \mathcal{E}(X + \Delta)$;
  $\rho = (\epsilon_1 - \epsilon_2)/(\Delta^T(\lambda\Delta + J^T\mathcal{E}(X + \Delta))$;
  if $\epsilon_2 \leq \epsilon_1$ **then**
    $X = X + \Delta$;
    $\lambda = \lambda \times \texttt{Max}(\frac{1}{3}, \texttt{Min}(1 - (2 \times \rho - 1)^3, \frac{2}{3}))$;
    $v = 2$;
    if $\epsilon_1 > 0.9999 \times \epsilon_2$ **then**
      stop;
    else
      $\epsilon_1 = \epsilon_2$;
    end
  else
    $\lambda = \lambda \times v$;
    $v = 2 \times v$;
  end
  $it = it + 1$;
end

primitives droites qui ne fixent qu'un seul degré de liberté. D'autre part, une primitive plus étendue comme une droite ou une courbe a plus de chance d'être partiellement occultée donc inutilisable.

Les détecteurs de points d'intérêt ont été introduits en 1970 par les travaux de Beaudet [26] puis de Moravec [27]. L'idée principale est d'extraire de l'image les points où le signal varie fortement. En effet, les zones de couleur unie apportent peu d'information, car un point de cette zone ne peut être précisément retrouvé dans l'image suivante. Harris et Stephen [28] proposent d'utiliser les points de l'image où les gradients verticaux et horizontaux varient fortement. Pour cela, ils mesurent la courbure de la fonction d'auto-corrélation du signal. Une courbure importante dans les deux directions signifie la présence d'un coin. Des travaux récents ont conduit à rendre ce détecteur plus robuste à l'illumination, mais aussi à tolérer des transformations affines. Une solution très utilisée dans le domaine de la vision par ordinateur est le détecteur SIFT pour *Scale Invariant Feature Transform* [29, 30] qui fournit d'excellents résultats en effectuant une analyse multi-échelle de la scène observée afin d'extraire des points d'intérêt. Toutefois, cela se fait au prix d'une quantité de calculs plus importante comparativement à la méthode de Harris et Stephen. En 2006, Rosten et Drummond ont proposé un détecteur nommé FAST pour *Features from Accelerated Segment Test* très utilisé pour sa rapidité d'exécution avec une précision acceptable.

1.6.1 Point d'intérêt de Harris

Le détecteur de Harris analyse la courbure de la fonction d'auto-corrélation du signal. Si en un point, cette courbure est importante dans une seule des directions, alors le point est situé sur un contour. Si les deux courbures sont faibles alors il s'agit d'une zone unie. Si les deux courbes sont importantes, alors le point est situé sur un coin. Le calcul de la fonction d'auto-corrélation est précédé d'un filtrage dont l'objectif est de diminuer le bruit photogrammétrique de l'image. Pour cela, un masque de convolution gaussien de dimension $\sigma \times \sigma$ est utilisé. σ est choisi empiriquement en fonction de l'image d'entrée. Puis on calcule la matrice symétrique de Harris en chaque pixel grâce à la relation :

$$M_H(x,y) = W * \begin{pmatrix} (\frac{\partial I}{\partial x})^2 & \frac{\partial I}{\partial x}\frac{\partial I}{\partial y} \\ \frac{\partial I}{\partial x}\frac{\partial I}{\partial y} & (\frac{\partial I}{\partial y})^2 \end{pmatrix} \tag{1.36}$$

Les dérivées partielles sont obtenues par dérivation numérique. Les valeurs propres λ_1 et λ_2 de cette matrice déterminent la nature du point observé.

1. si $\lambda_1 \approx 0$ et $\lambda_2 \approx 0$ alors le pixel (x, y) est dans une zone unie.
2. si $\lambda_1 \approx 0$ et λ_2 a une valeur importante, il s'agit d'un contour.
3. si λ_1 et λ_2 ont une valeur importante, alors le pixel est sur coin.

Puis, le critère de Harris est évalué par :

$$H(x,y) = \det(M_H) - k.\,\mathrm{trace}(M_H)^2 \tag{1.37}$$

avec un k un paramètre de réglage fixé dans nos applications à 0.04. Un point d'intérêt au sens de Harris est un maximum local du champ scalaire H. Soit (x, y) les coordonnées d'un point, alors si $H(x,y) \geq H(x+i, y+j)$ pour tout $(i,j) \in [-1,0,1]^2$ le point est un maximum local. Finalement, on conserve les plus grands maximums locaux comme points d'intérêt. Un exemple de détection est donné sur la figure 1.7.

1.6.2 Raffinement sous pixellique

Dans la méthode de Harris et Stephen, la détection des points d'intérêt est effectuée en coordonnées entières, donc au pixel près. Toutefois, l'utilisation d'un raffinement sous-pixellique permet d'augmenter la précision. Cela consiste à rechercher indépendamment sur chaque point d'intérêt de coordonnées

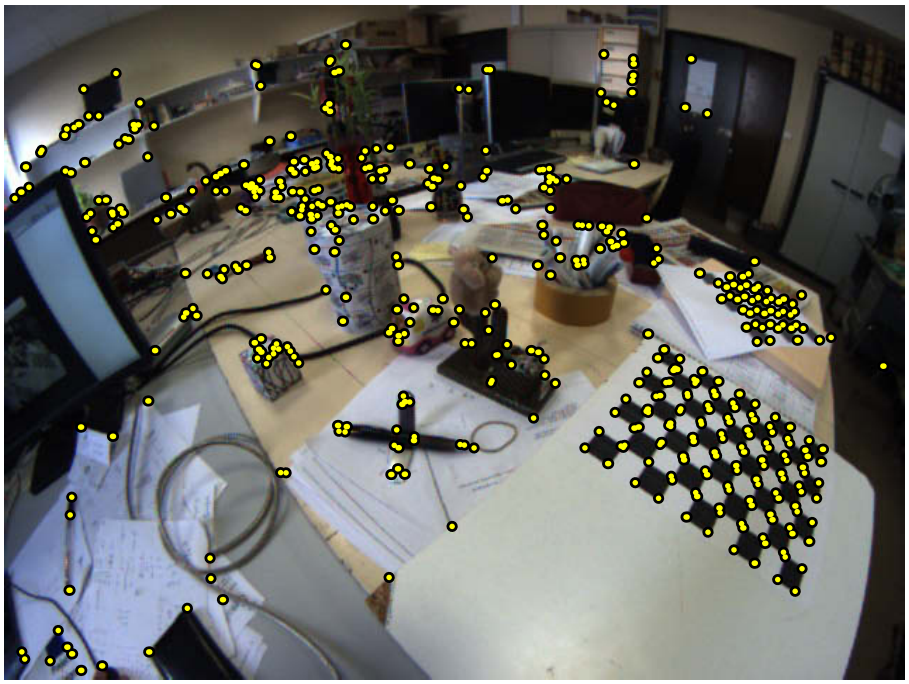


FIGURE 1.7 – Exemple de détection par la méthode de Harris. On remarque qu'on trouve des points d'intérêt (en jaune) principalement sur les coins et non sur les contours.

(x, y) un nouveau minimum x^* et y^*. On utilise pour cela deux polynômes de degré deux notés P_x et P_y afin d'interpoler le score de Harris de manière à ce que $P_x(x-1) = H(x-1, y)$, $P_x(x) = H(x, y)$ et $P_x(x+1) = H(x+1, y)$ et $P_y(x-1) = H(x, y-1)$, $P_y(x) = H(x, y)$ et $P_y(x+1) = H(x, y+1)$. Les nouvelles coordonnées x^* et y^* sont choisies comme les maximums de P_x et P_y. Une étude des apports du raffinement sous pixellique a été réalisée par Royer [31].

1.7 Calcul de descripteurs locaux

Afin d'effectuer un appariement des primitives détectées dans différentes images, un descripteur local de texture est extrait de l'image pour chaque primitive afin de caractériser celle-ci. Le descripteur SIFT [30] est connu pour sa robustesse, mais aussi pour ses nombreuses invariances. En effet, ce descripteur est invariant à l'échelle de la scène, à la rotation et aux transformations affines. Cependant, il nécessite une grande quantité de calculs ce qui le rend inutilisable dans des contextes d'applications temps réel. Le descripteur SURF [32] pour *Speeded Up Robust Features* présente les mêmes invariances et le temps de calcul plus acceptable permet son utilisation sur des plate-formes mobiles. Mais en fonction des séquences d'images, ses nombreuses invariances peuvent être un inconvénient. En environnement urbain par exemple, les coins des fenêtres sont souvent très similaires à une rotation près conduisant à l'appariement du coin bas gauche avec le coin bas droit de l'encadrure : le nombre de faux appariements peut alors dépasser celui de descripteurs plus basiques. Notre choix se porte sur une méthode plus simple basée sur des imagettes carrés. Pour cela, nous extrayons autour de chaque point une sous-image de taille $n \times n$ (en pratique $n = 11$) et nous utilisons le score ZNCC afin de les comparer. La ZNCC pour *Zero mean Normalized Cross Correlation* s'exprime comme une corrélation centrée normée dont l'expression en fonction des imagettes I_1 et I_2 est la suivante :

$$ZNCC(I_1, I_2) := \frac{\frac{1}{(2n+1)^2} \sum_{i=-n}^{n} \sum_{j=-n}^{n} \Big(I_1(i,j) - \overline{I}_1(n)\Big) \Big(I_2(i,j) - \overline{I}_2(n)\Big)}{\sigma_1(n) \cdot \sigma_2(n)} \tag{1.38}$$

Dans laquelle $\overline{I}_t(n)$ représente la moyenne du patch d'indice $t \in \{1, 2\}$ et vaut :

$$\overline{I}_t(n) := \frac{1}{(2n+1)^2} \sum_{i=-n}^{n} \sum_{j=-n}^{n} I_t(i,j) \tag{1.39}$$

Et $\sigma_t(n)$, l'écart type du patch d'indice $t \in \{1, 2\}$, qui a pour valeur :

$$\sigma_t(n) := \sqrt{\frac{1}{(2n+1)^2} \left(\sum_{i=-n}^{n} \sum_{j=-n}^{n} (I_t(i,j) - \overline{I}_t(n))^2 \right)} \tag{1.40}$$

Le descripteur ZNCC, qui ne possède que l'invariance à la luminosité, s'avère être une solution efficace dans le cadre du SLAM visuel. Dans nos travaux, nous utilisons principalement les descripteurs pour effectuer des appariements entre images consécutives sur un flux vidéo. Or, avec une fréquence d'acquisition suffisamment élevée (*ie.* supérieure à 10 images par seconde), l'apparence et la position des points d'intérêt entre deux images consécutives varient assez peu. Les descripteurs plus robustes tels que SIFT et SURF sont plus adaptés dans des contextes de reconstruction 3D à partir d'ensembles d'images désordonnés avec des points de vue très différents. De plus, la ZNCC offre des performances intéressantes en termes de temps de calcul grâce à l'utilisation d'instructions vectorielles [2] présentes sur la plupart des processeurs. La figure 1.8 illustre une utilisation de la ZNCC avec une méthode d'appariements de proche en proche où les descripteurs sont mis à jour dans chaque image intermédiaire (l'apport d'une telle méthode dans un algorithme de SLAM est présenté par Mouragnon [33]). Il est donc possible d'utiliser la ZNCC pour des applications de SLAM. C'est pourquoi, nous choisissons d'utiliser ce descripteur pour l'ensemble des travaux présentés dans le reste du manuscrit.

Récemment, des travaux se sont portés sur le développement de descripteurs binaires qui effectuent les calculs de corrélation directement sur les bits des pixels, sans utiliser d'opérations en virgule flottante. L'intérêt de ces approches réside dans l'utilisation des instructions binaires des processeurs qui sont plus performantes que les opérations en virgule flottante. Parmi les descripteurs binaires connus, on peut citer BRIEF [34], ORB [35] ou BRISK [36].

1.8 Géométrie épipolaire

La géométrie épipolaire modélise les relations liant le déplacement d'une caméra au déplacement de la projection des points de l'environnement dans l'image. Elle peut être utilisée entre deux images I_1 et I_2 dont les poses sont connues pour associer à un point ν_1 détecté dans I_1 une droite épipolaire dans I_2 : la droite épipolaire est définie par la projection dans l'image I_2 de la droite passant par le centre optique de l'image I_1 et de vecteur unitaire égal au lancer de rayon d'une détection de cette image. Si les détections ν_1 et ν_2 représentent la même primitive 3D dans deux images différentes alors la distance entre ν_2 et la droite épipolaire déterminée à partir de ν_1 devrait être faible (voir figure 1.9). Classiquement, la contrainte est appliquée à l'aide de la matrice fondamentale. La matrice fondamentale F est de taille 3×3 et relie, grâce à une relation matricielle, des points correspondants dans une paire d'images. En utilisant les coordonnées homogènes, si ν_1 et ν_2 sont des points homologues (ils correspondent au même point 3D) respectivement de l'image I_1 et I_2, alors ils doivent satisfaire la relation suivante :

$$\nu_2^T F \nu_1 = 0. \tag{1.41}$$

Avec

$$F = (K_1^T)^{-1} R \, [t]_\times K_2^{-1} \tag{1.42}$$

2. opérations arithmétiques exécutées en parallèle sur plusieurs flux de données scalaires

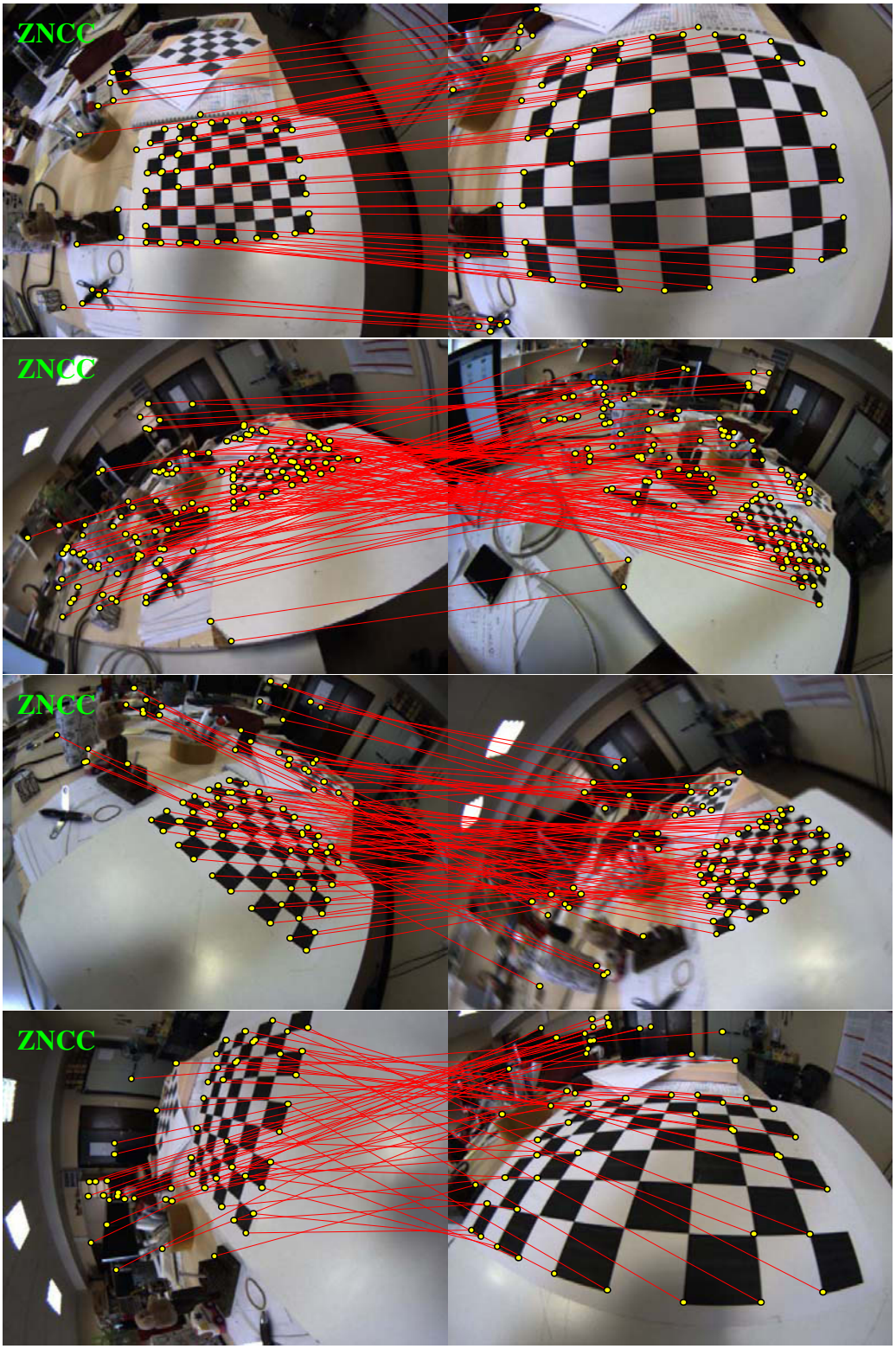


FIGURE 1.8 – Résultat d'appariements "de proche en proche" avec le descripteur ZNCC : l'apparence des descripteurs est mis à jour sur les images intermédiaires ce qui apporte une robustesse importante à la méthode.

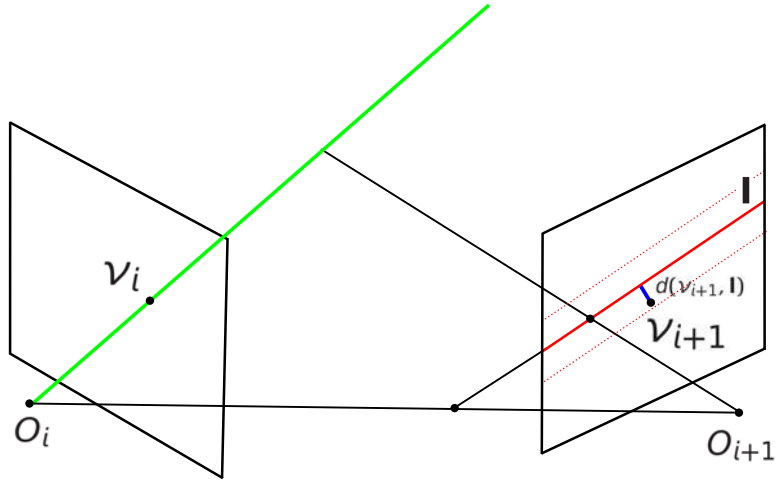


FIGURE 1.9 – La droite épipolaire $l_{\nu 1}$ en rouge relative à la détection ν_1 présente dans l'image I_1 définit dans l'image I_2 une zone de recherche. Seuls les points d'intérêt présents dans ce voisinage (dont ν_2 fait partie) vont être sélectionnés pour la comparaison des descripteurs.

Où K_1 et K_2 sont les matrices d'étalonnage intrinsèque respectivement de la première et de la seconde caméra. R et t représentent respectivement la rotation et la translation entre les poses des deux images. Dans ce cadre, l'équation de la droite épipolaire dans l'image I_2 pour un point d'intérêt ν_1 a pour expression :

$$l_{\nu_1} = F\nu_1 \tag{1.43}$$

l_{ν_1} est un vecteur de taille 3 contenant les paramètres de la droite épipolaire sous la forme $ax+by+c = 0$. On normalise ce résultat tel que $a^2 + b^2 = 1$ afin que le calcul de la distance d'un point d'intérêt de l'image I_2 avec la ligne soit le plus rapide possible. Cette distance s'écrit :

$$d(\nu_2, l_{\nu_1}) = \frac{|ax_{\nu_2} + by_{\nu_2} + c|}{\sqrt{a^2 + b^2}} = |ax_{\nu_2} + by_{\nu_2} + c| \tag{1.44}$$

Il est courant d'utiliser la contrainte épipolaire lorsque la position 3D des amers visuels est inconnue. Ceci est utilisé à la fois pour faciliter les méthodes d'appariement mais également pour la localisation :

- pour faciliter les appariements : la contrainte épipolaire permet de réduire la zone de recherche dans l'image I_2 (le long de la droite épipolaire) d'un point observé dans l'image I_1 afin d'éviter la comparaison de tous les descripteurs des deux images qui serait trop coûteuse en temps de calcul,
- pour la localisation : les distances entre les détections et les droites épipolaires associées peuvent être formalisées comme une somme d'erreurs dans une fonction de coût à minimiser avec un processus d'optimisation.

1.9 Triangulation : le point du milieu

La technique du point milieu consiste, à partir des deux détections dans deux images, à remonter aux rayons r_1 et r_2 par rétroprojection afin d'obtenir le segment NM orthogonal à r_1 et r_2. Le point triangulé P est alors choisi comme le milieu du segment NM. Par construction, NM est le segment reliant r_1 et r_2 pourvu de la longueur minimale. La figure 1.10 illustre l'obtention du point du milieu.

Soient o_1 et u_1 l'origine et le vecteur direction du rayon r_1, o_2 et u_2 l'origine et la direction du rayon r_2. Trouver le segment NM consiste à minimiser la distance notée D entre deux points de la

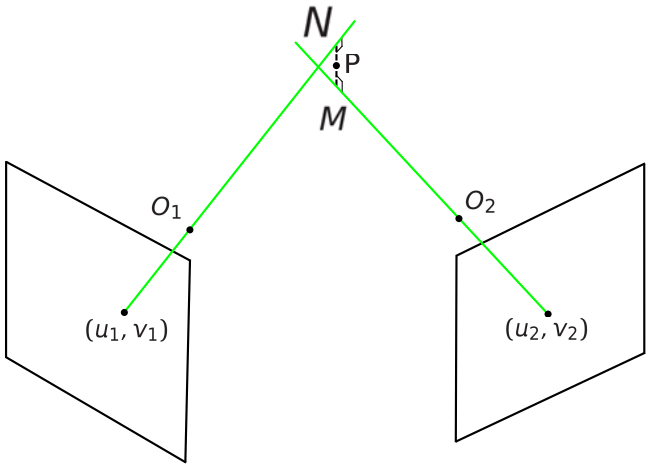


FIGURE 1.10 – Illustration de la méthode de triangulation du point milieu. Le point triangulé est choisi comme le milieu du segment minimal et orthogonal aux deux droites rétro-projetées.

droite :

$$D = \|o_1 + k_1 u_1 - (o_2 + k_2 u_2)\| = (o_1 + k_1 u_1 - o_2 - k_2 u_2)^T (o_1 + k_1 u_1 - o_2 - k_2 u_2) \tag{1.45}$$

où $k_1 \in \mathbb{R}$ et $k_2 \in \mathbb{R}$ représentent un paramétrage des droites. En dérivant cette formule par rapport à k_1 et k_2, on obtient le système d'équations suivant :

$$\frac{\partial D}{\partial k_1} = k_1 u_1^T u_1 - k_2 u_1^T u_2 + o_1^T u_1 - u_1^T o_2 \tag{1.46}$$

$$\frac{\partial D}{\partial k_2} = k_2 u_2^T u_2 - k_1 u_2^T u_1 + o_2^T u_2 - u_2^T o_1 \tag{1.47}$$

Enfin, on formalise ces équations scalaires sous la forme matricielle afin de produire l'expression suivante :

$$\begin{pmatrix} u_1^T u_1 & -u_1^T u_2 \\ -u_1^T u_2 & u_2^T u_2 \end{pmatrix} \begin{pmatrix} k_1 \\ k_2 \end{pmatrix} = \begin{pmatrix} -o_1^T u_1 + u_1^T o_2 \\ -o_2^T u_2 + u_2^T o_1 \end{pmatrix} \tag{1.48}$$

Ce qui permet de déterminer les valeurs des paramètres k_1 et k_2 par inversion. Le point P triangulé a donc pour expression :

$$P = \frac{o_1 + k_1 u_1 + o_2 + k_2 u_2}{2} \tag{1.49}$$

En présence d'un bruit important ou d'un déplacement insuffisant entre les poses, les rayons peuvent être divergents et donc le point triangulé se retrouve derrière les deux images. Nous supprimons ces aberrations en vérifiant que le point triangulé P est devant les poses caméra à l'aide de la condition suivante : $k_1 > 0$ et $k_2 > 0$.

1.10 RANSAC

L'algorithme nommé RANSAC (*RANdom SAmple Consensus*) a été proposé par Fischler et Bolles [37] pour résoudre des problèmes de calcul de pose à partir d'un ensemble de points 3D. Cet ensemble étant obtenu par mise en correspondance de points d'intérêt, il est possible qu'il y ait de nombreux faux appariements, donc de mauvais points 3D (dont l'erreur de reprojection est supérieure à un certain seuil). Le RANSAC est conçu pour résoudre ce problème malgré les données aberrantes, c'est pourquoi cette méthode est dite « robuste ».

Le RANSAC est applicable à de nombreux problèmes. De manière plus générale, il s'agit d'une méthode itérative utilisée pour estimer les paramètres d'un modèle à partir d'un ensemble d'observations contenant des *outliers*[3]. Cet algorithme se compose de plusieurs itérations de la séquence d'étapes suivante :

1. génération d'une solution à partir d'un tirage aléatoire d'un sous-échantillon de s observations,
2. calcul du nombre d'*inliers*[4] de la solution courante,
3. si le nombre d'*inliers* est le meilleur obtenu, alors la solution courante devient la solution de référence,
4. si le nombre d'itérations maximal est atteint ou si le nombre d'*inliers* de la solution de référence est supérieur à un seuil, une solution définitive est calculée à partir l'ensemble des *inliers* de la solution de référence (par optimisation par exemple) et l'algorithme est stoppé.

Le choix du nombre d'itérations N du RANSAC dépend de la proportion de faux appariements ϵ et du nombre d'observations s nécessaires pour générer une solution. Ainsi, il est possible d'estimer le nombre d'itérations N pour assurer au moins une génération de solution contenant un sous-échantillon uniquement constitué d'*inliers* avec une probabilité p (en général $p = 0.99$) :

$$N = \frac{\log(1-p)}{\log(1-(1-\epsilon)^s)} \tag{1.50}$$

1.11 Interpolation et approximation : les splines

Les splines sont utilisées pour interpoler et approximer des données en utilisant un modèle paramétrique de fonction. Dans le cas général, une spline est une fonction polynomiale par morceaux. Elle est définie par un degré d (correspondant au degré maximum des polynômes), un ensemble d'intervalles contigus ainsi qu'un polynôme sur chaque intervalle. Les intervalles sont définis par un ensemble de nœuds $t_0 < \cdots < t_n$ correspondant à $n+1$ réels strictement croissants. Une spline S est définie sur l'intervalle $[t_0, t_n]$ par :

$$S(t) = \sum_{i=0}^{d} p_{ij}(t-t_j)^i \text{ si } \begin{cases} t \in [t_j, t_{j+1}[\text{ et } j \in [0, n-2] \\ t \in [t_{n-1}, t_n] \end{cases}$$

où les valeurs p_{ij} pour $i \in [0, d[$ sont les coefficients du $j^{\text{ième}}$ polynôme. La contrainte de continuité de la spline est définie telle que la dérivée courbe à gauche d'un nœud soit identique à la dérivée de la courbe à droite :

$$\lim_{\substack{t \to t_i \\ t < t_i}} \frac{\partial^l s}{\partial t^l}(t) = \lim_{\substack{t \to t_i \\ t > t_i}} \frac{\partial^l s}{\partial t^l}(t) \qquad i \in [1, n-1], l \in [0, c_i - 1] \tag{1.51}$$

où $c_i \in [0, d-1]$ est la classe de continuité requise pour le nœud t_i (en général, on utilise $c_1 = d-1$).

1.11.1 Courbe de Bézier

L'idée des courbes de Bézier est d'utiliser des points de contrôle (qui sont approchés par la courbe) et non plus des points d'interpolation (par lesquels la courbe doit passer). On parlera alors d'approximations plutôt que d'interpolations. Une courbe de Bézier est définie par $n+1$ points de contrôle $P_0, \ldots, P_n$ telle que :

$$S(t) = \sum_{k=0}^{n} B_k^n(t) P_k \qquad \text{pour } 0 \leq t \leq 1 \tag{1.52}$$

3. Données « aberrante » ne satisfaisant pas un critère.
4. Données satisfaisant un critère.

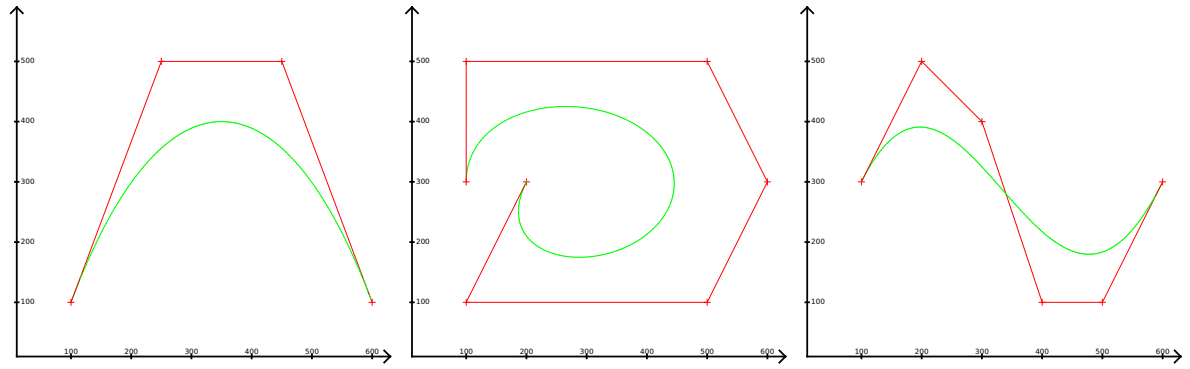


FIGURE 1.11 – Courbes de Bézier avec respectivement 4, 7 et 6 points de contrôle.

où les $B_k^n(t) = C_n^k t^k (1-t)^{n-k}$ sont les polynômes de base de Bernstein avec C_n^i les coefficients du binôme de Newton :

$$C_n^i = \frac{n!}{i!(n-i)!} \tag{1.53}$$

où k correspond au degré de la courbe et n est le rang du polynôme variant de 0 à n. Toutefois, le degré d'une courbe de Bézier est lié au nombre de points de contrôle, ce qui rend prohibitif le calcul d'une telle courbe dès que le nombre de points de contrôle devient important. La figure 1.11 illustre quelques exemples de courbes de Bézier.

1.11.2 B-spline

L'idée des B-splines est de remplacer les polynômes de Bernstein par des fonctions de base. C'est pourquoi, les B-splines présentent les avantages des courbes de Bézier mais sans les inconvénients. Par exemple, le degré d'une B-spline est fixe (et non plus lié au nombre de points de contrôle). Une B-spline est définie par $n+1$ points de contrôle $p_0, \ldots, p_n$ ainsi qu'un ensemble de réels $t_0 \leq \cdots \leq t_m$, avec $m \geq n+k$, et les fonctions de base $B_{i,k}$ de degré k associées. On définit une B-spline par :

$$S(t) = \sum_{i=0}^{n-1} B_{i,k}(t) p_i, \qquad \text{pour } t_k \leq t < t_n \tag{1.54}$$

et les fonctions de base $B_{i,k}(t)$ sont calculées en utilisant la formule récursive de Cox-De Boor [38, 39] :

$$\begin{cases} 0 \leq i \leq m-1 \\ B_{i,0}(t) = 1 \quad \text{si } t \in [t_i, t_{i+1}[, \quad B_{i,0}(t) = 0 \text{ sinon} \end{cases} \tag{1.55}$$

$$\begin{cases} k \geq 1 \text{ et } 0 \leq i \leq m-k-1 \\ B_{i,k}(t) = \dfrac{t-t_i}{t_{i+k}-t_i} B_{i,k-1}(t) + \dfrac{t_{i+k+1}-t}{t_{i+k+1}-t_{i+1}} B_{i+1,k-1}(t) \end{cases} \tag{1.56}$$

Un point de la courbe $S(t)$ ne dépend que des $k+1$ points de contrôle tel que $t_j \leq t \leq t_{j+1}$. Réciproquement, un point de contrôle P_j n'a d'influence que sur l'intervalle $[t_j, t_{j+k+1}]$ de la courbe. Ceci offre un contrôle local de la B-spline.

B-spline cubique uniforme

Les B-splines cubiques uniformes, notées UCBS pour *Uniform Cubic B-Spline*, sont un cas particulier des B-splines et sont intéressantes du fait de leur simplicité et de leur efficacité. Le degré 3 de ces splines fait qu'elles sont à la fois flexibles et nécessitent relativement peu de calculs. La dérivabilité

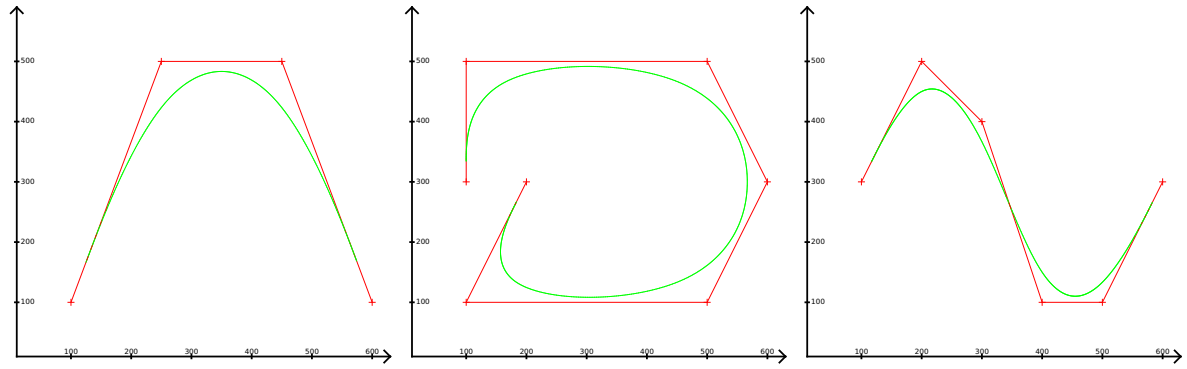


FIGURE 1.12 – B-Splines cubiques uniformes avec respectivement 6, 9 et 8 points de contrôle (les nœuds de début et fin sont doublés).

à l'ordre 2 impose l'égalité des dérivées premières et secondes aux points de raccordement des polynômes. La propriété « uniforme » signifie que les nœuds sont uniformément répartis. Les fonctions de base des UCBS sont les suivantes :

$$B_i(t) = \begin{cases} b_3(t) = t^3/6 & \text{si } t \in [t_i, t_{i+1}[\\ b_2(t) = (-3t^3 + 3t^2 + 3t + 1)/6 & \text{si } t \in [t_{i+1}, t_{i+2}[\\ b_1(t) = (3t^3 - 6t^2 + 4)/6 & \text{si } t \in [t_{i+2}, t_{i+3}[\\ b_0(t) = (-t^3 + 3t^2 - 3t + 1)/6 & \text{si } t \in [t_{i+3}, t_{i+4}[\\ 0 & \text{sinon.} \end{cases} \tag{1.57}$$

L'évaluation de cette spline en un point t est le résultat du mélange des 4 points de contrôle $p_i, p_{i+1}, p_{i+2}, p_{i+3}$ correspondant aux réels $t_i, t_{i+1}, t_{i+2}, t_{i+3}$. Ces points de contrôle sont choisis tels que $t_{i+1} \leq t < t_{i+2}$. Le paramètre t est normalisé entre p_{i+1} et p_{i+2} (on note qu'en utilisant une B-spline uniforme, $(t_2 - t_1)$ est constant) : $u = (t - t_{i+1})/(t_{i+2} - t_{i+1})$. Avec l'abscisse normalisé u, la position $S(u)$ est obtenue par :

$$S(u) = p_i + (p_{i+1} - p_i)B_1(u) + (p_{i+2} - p_{i+1})B_2(u) + (p_{i+3} - p_{i+2})B_3(u) \tag{1.58}$$

La figure 1.12 illustre quelques exemples de B-splines cubiques uniformes.

B-spline cubique uniforme cumulée

Kim *et al.* [40] ont proposé une forme cumulée des fonctions de base des UCBS telle que les coefficients vus dans l'équation 1.57 soient sommés (cumulés) :

$$\tilde{B}_{i,k}(t) = \sum_{j=i}^{n} B_{j,k}(t) \tag{1.59}$$

Il en résulte les fonctions de base suivantes :

$$\tilde{B}_i(t) = \begin{cases} \tilde{b}_3(t) = 1.0 & \text{si } t \in [t_i, t_{i+1}[\\ \tilde{b}_2(t) = (t^3 - 3t^2 + 3t + 5)/6 & \text{si } t \in [t_{i+1}, t_{i+2}[\\ \tilde{b}_1(t) = (-2t^3 + 3t^2 + 3t + 1)/6 & \text{si } t \in [t_{i+2}, t_{i+3}[\\ \tilde{b}_0(t) = t^3/6 & \text{si } t \in [t_{i+3}, t_{i+4}[\\ 0 & \text{sinon.} \end{cases} \tag{1.60}$$

Le principal avantage de cette forme cumulée est d'offrir des dérivés première et seconde relativement simples. La position $S(u)$ au temps normalisé u est calculable avec :

$$S(u) = p_i + (p_{i+1} - p_i)\tilde{B}_1(u) + (p_{i+2} - p_{i+1})\tilde{B}_2(u) + (p_{i+3} - p_{i+2})\tilde{B}_3(u) \tag{1.61}$$

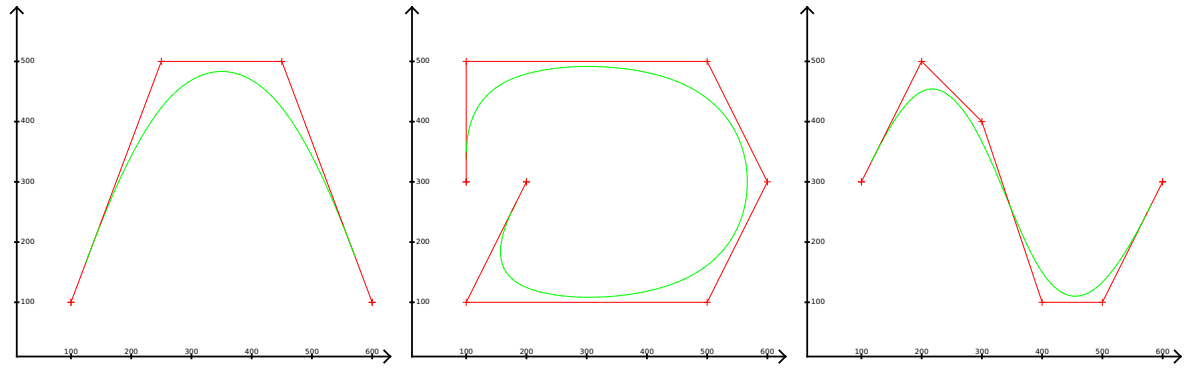


FIGURE 1.13 – B-Splines cubiques uniformes cumulées avec respectivement 6, 9 et 8 points de contrôle (les nœuds de début et fin sont doublés).

La figure 1.13 illustre quelques exemples de B-splines cubiques uniformes cumulées. Les B-splines rationnelles non uniformes, plus connues sous le nom de NURBS (*Non Uniform Rational Basis Splines*), sont une généralisation des B-splines et ne seront pas présentées ici.

1.12 Programmation

Les travaux présentés dans cette thèse sont fortement liés à l'implémentation efficace d'algorithmes. Pour appréhender les chapitres suivants, et notamment la partie II sur l'implémentation d'une bibliothèque d'optimisation, cette section présente certains concepts de programmation (héritage et polymorphisme) ainsi que les notions d'implémentations abstraites et spécialisées pour finir sur la méta-programmation.

1.12.1 Héritage

L'héritage est un concept de la programmation orientée objet consistant à ré-utiliser les propriétés d'une classe dite « mère » dans une ou plusieurs classes dérivées dites « filles ». Ces membres peuvent être des méthodes (fonctions), des attributs (variables) ou des types encapsulés dans la classe mère. Dans l'exemple ci-dessous, la classe **Fille** hérite des propriétés de la classe **Mere**.

```
struct Mere
{
  int compute(int b, int c) { return b + c; }
  int a;
  using Pair = std::pair<int,int>;
};

struct Fille : Mere
{
};

// instance de la classe Fille :
Fille fille;

//utilisation possible de fille.a, Fille::Pair, fille.compute(...)
```

1.12.2 Polymorphisme

Le polymorphisme est un mécanisme permettant, via l'héritage, de redéfinir dynamiquement (pendant l'exécution du programme) dans les classes dérivées les comportements qui sont implémentés dans la classe mère dite abstraite. La classe mère définit alors un comportement par défaut pour chacune des méthodes, et les classes filles peuvent redéfinir l'implémentation de ces méthodes. Avec le langage C++, ce mécanisme est réalisé avec le mot clé **virtual** devant les fonctions membres à spécialiser. Cela signale au compilateur qu'à l'exécution du programme, il existera potentiellement une version spécialisée de cette fonction dans les classes filles. Considérons les classes suivantes :

```
struct Mere
{
  virtual int compute(int b, int c) { return b + c; }
};

struct Fille1 : Mere
{
  int compute(int b, int c) { return b * c; }
};

struct Fille2 : Mere
{
  int compute(int b, int c) { return b - c; }
};
```

L'utilisation du polymorphisme permet, via l'abstraction offerte par la classe **Mere**, un changement dynamique de comportement de l'instance **mere** à la classe **Fille1** puis à la classe **Fille2** :

```
std::unique_ptr<Mere> mere(new Mere);
a = mere->compute(b,c);// a = b + c
mere.reset(new Fille1);
a = mere->compute(b,c);// a = b * c
mere.reset(new Fille2);
a = mere->compute(b,c);// a = b - c
```

L'utilisation de la classe **unique_ptr** permet ici une gestion automatique de la mémoire pour les instances qui ne sont plus référencées.

Inconvénient du polymorphisme

Le polymorphisme est un concept très puissant permettant une grande modularité pour le développement d'applications complexes. Ce mécanisme est intensivement exploité dans des bibliothèques comme Qt [5] où toutes les classes appartiennent à une même hiérarchie et héritent donc d'un même type de base. Toutefois, dans certains cas, l'utilisation du polymorphisme nuit aux performances. Cela est dû à causes :

1. Le comportement d'une classe abstraite étant défini dynamiquement, il ne peut pas être prédit à la compilation. C'est pourquoi, lorsqu'une méthode virtuelle (utilisant le polymorphisme) est appelée, le programme doit rechercher, parmi les différentes définitions de cette fonction dans les classes dérivées, celle qui est la plus spécialisée. Ce calcul d'adresse nécessite un certain nombre d'instructions processeur (ce nombre dépend de nombreux facteurs).

5. http://www.qt.io/

2. Lorsque cela est possible, le compilateur utilise l'*inlining* de fonction. Ce mécanisme consiste à remplacer, dans le code exécutable, l'appel de la fonction par le corps de cette même fonction. Cela permet de supprimer un appel de fonction à l'exécution et donc d'améliorer les performances. Bien entendu, ce mécanisme n'a d'intérêt que s'il y a peu de traitement dans le corps de la fonction et si celle-ci est souvent appelée, sinon le gain est négligeable. Dans le cas de fonctions virtuelles, il n'est pas possible de déterminer à la compilation quelle spécialisation sera utilisée à l'exécution, ce qui rend impossible l'utilisation de l'*inlining*.

C'est pourquoi, il faut éviter l'utilisation du polymorphisme sur les fonctions qui sont souvent appelées, d'autant plus si le nombre d'instructions effectuées dans le corps de ces fonctions est faible comparativement au calcul d'indirection généré par le polymorphisme.

1.12.3 Implémentations abstraites et spécialisées

Cette sous-section définit les notions d'implémentations « abstraites » et « spécialisées » qui seront ré-utilisées dans la suite du manuscrit, notamment dans la partie II.

Dans ce manuscrit, on appelle « implémentations abstraites » des algorithmes écrits dans un langage de programmation haut niveau privilégiant, la lisibilité du code, la modularité et la portabilité, aux performances d'exécution. Pour cela, ces implémentations ont recours aux propriétés dynamiques et d'abstractions du langage (*i.e.* allocation dynamique, héritage, polymorphisme, ...), et limitent l'utilisation d'instructions bas niveaux (*i.e.* liées à l'architecture matérielle) ou d'*a priori* trop forts sur la nature du problème. Par opposition, on parlera d'« implémentations spécialisées » pour les algorithmes où le code source est spécifiquement écrit afin de résoudre efficacement un problème en particulier. Pour cela, deux voies sont explorées : l'implémentation spécifique de l'algorithme basée sur les propriétés intrinsèques du problème connues au moment de l'écriture du programme, et la spécialisation du code pour l'architecture matérielle cible. C'est pourquoi, à défaut d'être performantes, les implémentations abstraites sont en général modulaires et faciles à utiliser (avec une syntaxe intuitive). Tandis que les implémentations spécialisées, bien qu'efficaces, souffrent d'un portage plus difficile, d'une syntaxe souvent compliquée, et sont en générales restreintes à la résolution de problèmes spécifiques.

Considérons à titre d'exemple un algorithme qui calcule la somme, élément par élément, de 3 tableaux (**a**,**b**,**c**) entre eux et stocke le résultat dans un 4$^{\text{ème}}$ noté **s**. Chaque tableau est composé de 4 valeurs flottantes implémenté en C++ de la façon suivante :

```
#include <vector>
using Floats = std::vector<float>;

Floats a {1,2,3,4};
Floats b {5,6,7,8};
Floats c {9,10,11,12};
Floats s {0,0,0,0};
```

Une première solution, privilégiant la clarté du code, consiste à définir un opérateur binaire + qui prend deux tableaux en entrée et renvoie la somme calculée élément par élément :

```
Floats operator+(Floats x, Floats y)
{
  Floats result(x.size());// allocation
  for(size_t i = 0 ; i < x.size() ; ++i)
    result[i] = x[i] + y[i];
  return result;
}
```

Ainsi, il est possible de résoudre le problème avec une écriture simple et intuitive :

```
s = a + b + c;
```

De plus, ce code est assez flexible : il est possible de chaîner les sommes $(a + b + c + d + \dots)$ et l'implémentation fonctionne quel que soit le nombre d'éléments dans les tableaux. Ainsi, par abus de langage, on parlera d'implémentation abstraite car ce code est relativement modulaire. Toutefois, chaque utilisation de l'opérateur + génère un tableau temporaire qui doit être alloué, affecté, copié et supprimé, ce qui nuit aux performances. Pour résoudre cet inconvénient, il est préférable d'utiliser l'implémentation suivante :

```
for(size_t i = 0 ; i < a.size() ; ++i)
  s[i] = a[i] + b[i] + c[i];
```

Cette approche est relativement performante, car elle n'utilise pas de variables temporaires, mais l'implémentation est plus verbeuse que la précédente. Il est encore possible d'améliorer les performances en utilisant des *a priori* sur la nature du problème. Par exemple, on sait qu'il y a 4 valeurs par tableau. On peut donc dérouler manuellement la boucle **for** afin de supprimer le calcul de l'indice **i** ainsi que les sauts effectués à chaque itération dans la pile d'exécution du programme :

```
s[0] = a[0] + b[0] + c[0];
s[1] = a[1] + b[1] + c[1];
s[2] = a[2] + b[2] + c[2];
s[3] = a[3] + b[3] + c[3];
```

Cette implémentation, bien que plus performante, est moins générique. En effet, cela ne fonctionne qu'avec des tableaux de 4 éléments. En poussant le raisonnement plus loin, il est possible d'utiliser les instructions vectorielles des processeurs pour effectuer les 4 lignes de calcul simultanément grâce au SIMD : Single Instruction Multiple Data. Avec le jeu d'instructions spécifique aux architectures x86, l'implémentation prend alors cette forme :

```
_mm_store_ps(
  s.data(),
    _mm_add_ps(_mm_load_ps(a.data()),
      _mm_add_ps(_mm_load_ps(b.data()),
        _mm_load_ps(c.data()))));
```

Ce code est le plus performant des 4 implémentations présentées, en contrepartie c'est également le moins lisible. De plus, cette solution fonctionne uniquement pour calculer la somme de 3 vecteurs contenant 4 valeurs flottantes de 32 bits alignées en mémoire sur une architecture de type x86. Ainsi, ce code est complètement spécialisé pour la résolution d'un problème en particulier. Cette spécialisation s'est faite sur les données du problème connu au moment de l'écriture du code : le calcul à effectuer, la dimension des tableaux et l'architecture matérielle. C'est pourquoi, on qualifie cette implémentation de spécialisée.

Il est difficile d'allier abstraction et spécialisation. Toutefois, des travaux récents tentent de réunir les avantages des deux approches afin de proposer des solutions simples à utiliser et exploitant au mieux les données connues à la compilation ainsi que les architectures matérielles modernes. Par exemple, on voudrait pouvoir écrire le code

```
s = a + b + c;
```

et obtenir les performances d'un code utilisant le SIMD. Cela est rendu possible avec des bibliothèques comme NT2 [41] en utilisant des concepts de méta-programmation dont quelques idées sont abordées dans la section suivante.

1.12.4 La méta-programmation

La méta-programmation désigne l'écriture de méta-programmes, autrement dit, des programmes qui en écrivent d'autres. Ces programmes peuvent être de la forme d'une bibliothèque proposant un niveau d'abstraction élevé et un mécanisme d'optimisation *ad hoc*, on parle alors de « bibliothèque active » [42]. De telles bibliothèques procèdent à la fois à des optimisations haut niveau du code (exploitation des *a priori* disponibles correspondant aux données statiques) et des optimisations bas niveau usuelles via le compilateur. Czarnecki *et al.* parlent de programmation générative dans [43] définie comme « un paradigme de développement logiciel compréhensible pour obtenir intentionnellement du code ré-utilisable et adaptatif sans faire de compromis sur le temps d'exécution et les ressources computationnelles du logiciel produit ». L'idée sous-jacente de ces méthodes est de définir, non pas un programme pour résoudre un problème, mais un programme qui va écrire un programme pour résoudre un problème.

En C++, la méta-programmation est basée sur l'utilisation des types génériques *templates* qui forment un langage Turing complet. Autrement dit, il est possible de générer toutes les fonctions calculables, au sens de Turing, à travers l'utilisation des *templates* C++ et d'une écriture potentiellement non-triviale. Toutefois, l'évolution du langage C++ à travers de nouvelles normes[6] (C++ 11, C++ 14, C++ 17) permet l'utilisation de mécanismes simplifiant considérablement l'écriture de la méta-programmation *template*, grâce notamment aux arguments et *templates* variadiques, les mots clés **auto** et **decltype**, et les fonctions *lambda*.

Des exemples de calcul à la compilation et de moteur d'expressions sont présentés pour illustrer comment détourner le compilateur de sa tâche originale pour le forcer à effectuer des calculs ou ré-interpréter un code.

Calcul à la compilation

Il est possible de forcer le compilateur à effectuer des calculs. Prenons comme exemple un code source simple additionnant des instances de type **Int** contenant une valeur entière nommée **value**. NB : dans les codes qui suivent, les fonctions du C++ **assert** et **static_assert** renvoient une erreur si l'égalité n'est pas vérifiée. Une implémentation possible pour une résolution dynamique (à l'exécution du programme) de l'addition est donnée par le code suivant :

```
#include <cassert>

struct Int { int value;};

Int operator+(Int i, Int j) { return Int{i.value + j.value};}

int main(){
  Int i{1};
  Int j{2};
  assert( (i+j+i+j).value == 6 );
}
```

Le calcul est effectué à l'exécution du programme alors que toutes les informations sont disponibles dans le code source. Une des idées de la méta-programmation est d'utiliser ces informations pour effectuer le calcul dès le processus de compilation. Plutôt que de modifier le compilateur, on utilise les *templates* C++ qui sont des types génériques résolus à la compilation. Ce second code

```
template<int I> struct Int { enum { value = I }; };
```

6. https://isocpp.org/std/the-standard

```
template<int I, int J> Int<I+J> operator+(Int<I>, Int<J>) { return {};}

int main(){
  Int<1> i;
  Int<2> j;
  static_assert( decltype(i+j+i+j)::value == 6, "");
}
```

résout le même problème mais la syntaxe particulière à base de *templates* force l'évaluation du résultat à la compilation. Ainsi, aucun calcul n'est effectué à l'exécution, le second programme contient moins d'instructions et sera plus performant que le premier.

Moteur d'expressions

La méta-programmation est également utilisée afin que le compilateur ré-interprète le code source. Pour cela, on utilise un moteur d'expressions. Il s'agit de surcharger les opérateurs de base du C++ afin de créer un nouveau langage. Le code source utilisant ce langage va générer, pendant la compilation, des expressions représentées par un arbre syntaxique. Lorsque que le résultat est requis, souvent à travers l'opérateur d'affectation, l'expression est évaluée. Cela permet d'effectuer un pré-traitement sur le code source, afin d'appliquer des optimisations qui sont propres au domaine d'application, et qui ne peuvent pas être connues par les compilateurs. Il existe différentes approches pour réaliser des moteurs d'expressions. L'une d'entre elles consiste à utiliser le polymorphisme statique (même principe que le polymorphisme vu précédemment mais appliqué à des types résolus à la compilation) pour définir un langage où les opérateurs et les données héritent d'un même type. Cela permet de lever les ambiguïtés sur les surcharges d'opérateurs tout en conservant la possibilité de convertir statiquement les éléments du langage. Par exemple, on souhaite écrire une somme de tableaux (comme l'exemple vue dans la section 1.12.3) avec le code suivant :

```
Floats a = b + c + d
```

Toutefois, pour des soucis de performances, on veut éviter les variables temporaires et donc avoir un comportement équivalent au code ci-dessous :

```
for(size_t i = 0 ; i < a.size() ; ++i)
  s[i] = a[i] + b[i] + c[i];
```

Pour cela, on définit un nouveau langage en utilisant un moteur d'expressions afin de modifier l'interprétation de l'opérateur + par le compilateur. Ce problème est résolu par le code 1.1.

Code 1.1 – Moteur d'expressions pour l'addition de vecteur

```
1  #include <vector>
2
3  template<class T> struct Expr { T const & cast() const {
4    return static_cast<T const &>(*this); }};
5
6  struct Floats : Expr<Floats> {
7    std::vector<float> floats;
8    Floats(std::initializer_list<float> init):floats(init){}
9    std::size_t size() const { return floats.size(); }
10   float operator[](std::size_t i) const { return floats.at(i); }
11
12   Floats(auto const & xpr) {
13     for(std::size_t i = 0 ; i < xpr.size() ; ++i)
```

```
14      floats.push_back(xpr[i]);
15    }
16  };
17
18  template<class Xpr1, class Xpr2> struct Add : Expr<Add<Xpr1,Xpr2>>{
19    Xpr1 const & xpr1;
20    Xpr2 const & xpr2;
21    Add(Xpr1 const & xpr1_, Xpr2 const & xpr2_):xpr1(xpr1_),xpr2(xpr2_){}
22    std::size_t size() const { return xpr1.size(); }
23    auto operator[](std::size_t i) const { return xpr1[i] + xpr2[i]; }
24  };
25
26  template<class Xpr1, class Xpr2>
27  Add<Xpr1,Xpr2> operator+(Expr<Xpr1> const& xpr1, Expr<Xpr2> const& xpr2){
28    return {xpr1.cast(),xpr2.cast()};
29  }
30
31  int main(){
32    Floats a {1,2,3,4};
33    Floats b {5,6,7,8};
34    Floats c {9,10,11,12};
35    Floats r = a + b + c;
36  }
```

Chaque élément du langage (opérateur + et le type **Floats**) hérite du type **Expr**. Les expressions du langage sont donc composées d'éléments de type **Expr**. De plus, l'opérateur + est redéfini pour construire un élément du langage correspondant à l'opérateur binaire d'addition sous la forme de la structure **Add**. Cette structure possède deux entrées **Xpr1** et **Xpr2** qui sont des éléments du langage, donc potentiellement de type **Floats** ou **Add**. Ainsi, l'expression a + b + c produira l'arbre syntaxique **Add**(a,**Add**(b,c)) dont le calcul sera effectué et affecté à la ligne 35 via le constructeur de copie de la classe **Floats** ligne 12.

Les moteurs d'expressions sont utilisés dans de nombreuses bibliothèques matricielles pour améliorer les performances. De plus, des bibliothèques comme Boost.Proto facilitent l'utilisation de telles approches afin de définir des langages (à travers le C++) spécifiques à un domaine d'application (algèbre linéaire, traitement d'images, calculs de dérivées automatique, ...). Le résultat est une syntaxe intuitive pour l'utilisateur tout en conservant de bonnes performances en temps de calculs.

Première partie

SLAM visuel

Le SLAM (*Simultaneous Localization And Mapping*) est une technique de SfM (*Structure from Motion*) consistant à localiser un système de perception (*e.g.* une caméra) par rapport à une représentation de l'environnement, couramment appelée carte, continuellement mise à jour. Cette approche permet l'exploration d'environnements inconnus. La carte est régulièrement actualisée pour intégrer les nouveaux éléments perçus, et la pose du système est actualisée en comparant les données observées avec les données déjà présentes dans la carte. Ainsi, les processus de localisation et de mise à jour de la carte sont effectués simultanément mais de manière dépendante : la localisation utilise le résultat de la cartographie qui a nécessité la connaissance de la pose courante pour géo-référencer les nouvelles données dans la carte existante.

Les méthodes de SLAM et plus généralement de reconstruction 3D peuvent être employées dans de nombreux contextes applicatifs. Afin de situer nos travaux dans la diversité des variantes de SLAM, on dresse ici un panorama non exhaustif illustré par la figure 1.14. Les capteurs peuvent être de différentes natures, comme des lasers, des capteurs de profondeur, ou des caméras. Il existe différentes familles de caméras : vidéo, infra-rouge, temps de vol... Parmi les caméras vidéos, certaines sont *global shutter* (l'acquisition de l'ensemble de l'image se fait simultanément) et d'autres sont *rolling shutter* (chaque ligne de l'image est prise à un temps différent). Certaines méthodes nécessitent que les paramètres intrinsèques de la caméra soient précisément connus. Il existe des méthodes de reconstruction en environnement déformable et d'autres en environnement rigide. Les méthodes de localisation peuvent s'effectuer en environnement connu, à partir d'un modèle, ou en environnement inconnu (sans modèle) en utilisant la géométrie épipolaire entre les images consécutives du flux vidéo. Certaines méthodes supposent que le mouvement entre deux images successives est relativement faible. D'autres supportent un changement de point de vue très important entre différentes prises d'images d'un même environnement. Certaines méthodes utilisent l'intégralité des pixels de l'image pour effectuer une reconstruction dense tandis que d'autres approches sont dites éparses. Ces dernières se focalisent sur l'information utile contenue dans l'image (les points de contour correspondant aux gradients importants) afin de limiter les calculs. De plus, l'estimation des paramètres nécessaires pour la localisation peut s'effectuer avec des méthodes probabilistes ou par optimisation (moindres carrés non linéaires).

Parmi toutes les configurations possibles et les hypothèses requises, il est nécessaire de faire des compromis et de se positionner sur une problématique. Que ce soit pour la robotique mobile ou la Réalité Augmentée, la qualité de la localisation est plus importante que la qualité du modèle d'environnement reconstruit. De plus, la méthode doit pouvoir être déployée facilement, sur des appareils de la vie courante tels que les ordinateurs personnels ou les *smartphones*. Cela impose des contraintes en terme de capacité de calcul et limite les capteurs potentiels à ceux présents sur ce type d'architecture. C'est pourquoi, nous traitons ici le cas particulier (illustré en rouge sur la figure 1.14) de la localisation d'une caméra vidéo *global shutter* calibrée dans un environnement rigide inconnu par analyse en continu du flux vidéo sur un ordinateur personnel grand public.

Cette partie se compose d'un premier chapitre avec un survol des méthodes de SLAM par vision monoculaire et présente une implémentation temps-réel réalisée au cours de ce travail de thèse. Des expérimentations mettant en évidence les performances de l'algorithme sont présentées en fin de partie.

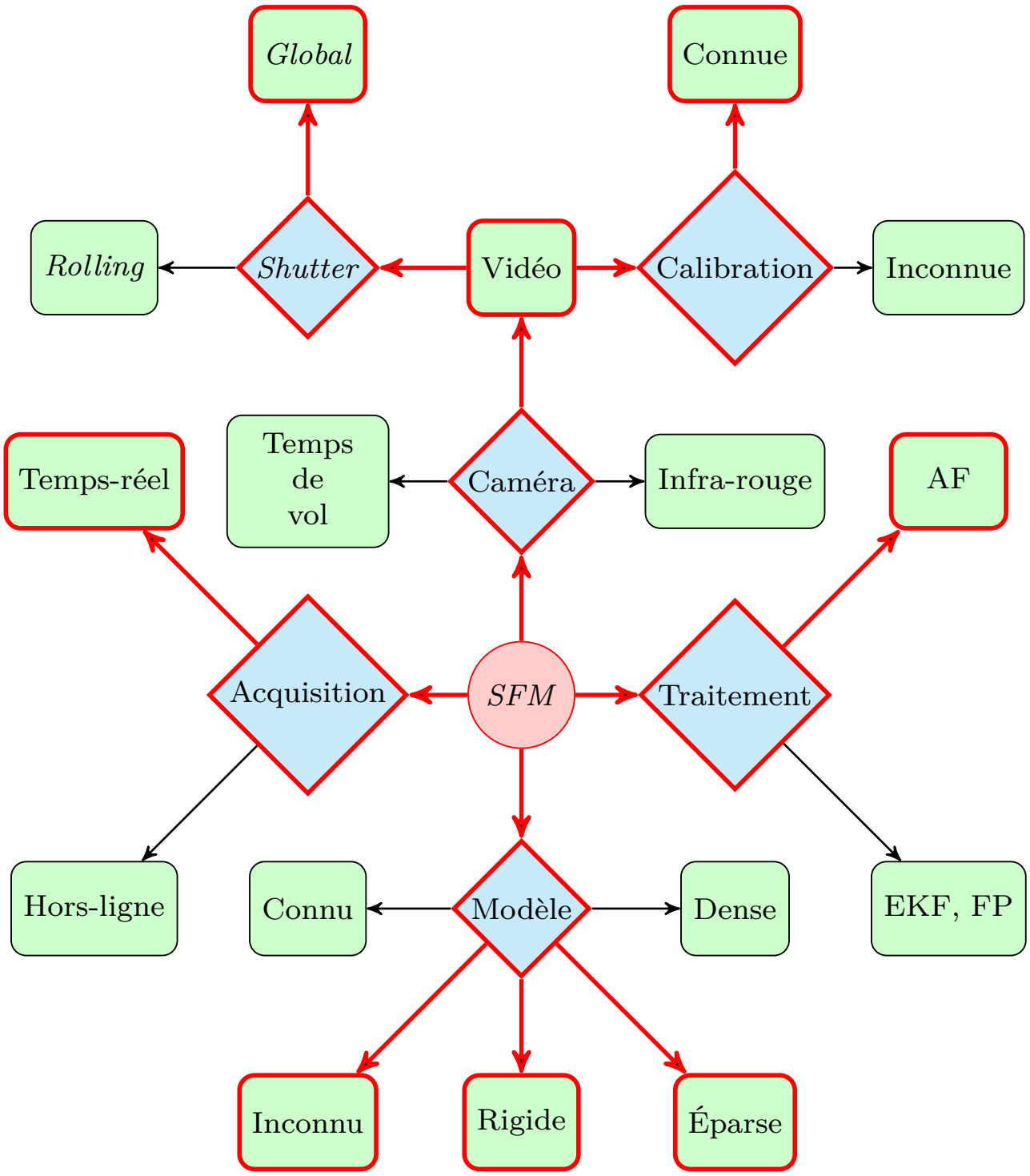


FIGURE 1.14 – Quelques variantes des méthodes de *Structure From Motion* (SFM) par caméra (EKF : filtre de Kalman étendu, FP : filtre à particules, AF : Ajustement de faisceaux, *Shutter* : mode d'exposition des caméras vidéos). La méthode utilisée dans nos travaux est mise en évidence en rouge (traits épais).

Chapitre 2

Présentation du SLAM visuel

Les méthodes de type SLAM visuel (*Visual Simultaneous Localization And Mapping*) sont des techniques de SfM (*Structure from Motion*). L'objectif est de calculer le déplacement d'une caméra en analysant le flux vidéo. Une première étape consiste à localiser la caméra en estimant la pose dans un espace 3D arbitraire par mise en correspondance des informations contenues dans les images avec le modèle d'environnement connu. La seconde étape a pour but de mettre à jour le modèle de l'environnement avec les nouvelles informations contenues dans les images. Ces algorithmes sont utiles dans de nombreuses applications comme la robotique mobile ou la réalité augmentée. On distingue deux types d'approches pour résoudre les problèmes de SLAM : les méthodes probabilistes utilisant des filtres Bayésien récursifs (filtres de Kalman, filtres à particules,...) et les méthodes d'optimisation de type ajustement de faisceaux (AF). Le lecteur intéressé pourra se référer aux travaux de Salas-Moreno [44] pour un historique détaillé des méthodes de SLAM. Ce chapitre se compose de deux sections. La première présente les approches probabilistes, puis une seconde présente les approches par AF.

2.1 SLAM probabiliste

Les premiers travaux en matière de SLAM probabiliste ont été réalisés par Smith et Chesseman [45] en 1988. Ils ont proposé d'estimer de façon conjointe la position des amers visuels de l'environnement et la trajectoire en utilisant un filtre de Kalman étendu (EKF). Ces bases théoriques ont ensuite été développées par Moutarlier et Chatila [46] pour des applications à base de lasers et de sonars, et sont toujours utilisées dans les implémentations modernes de SLAM probabiliste [47, 48, 49, 50, 51, 52]. L'un des premiers systèmes complets de SLAM EFK utilisant la vision est développé par Davison *et al* en 1998 [49] pour localiser un robot se déplaçant sur un plan. Ils utilisent la vision active pour focaliser la caméra sur les amers visuels et ainsi reconstruire une carte 3D. En 2003, Davison *et al* [53] proposent une implémentation temps-réel d'un SLAM visuel par EKF sur un ordinateur grand public avec une caméra portée à la main.

Toutefois, les méthodes probabilistes sont limitées par les temps de calcul trop importants lorsque la taille de la carte augmente. La complexité algorithmique de la mise à jour est de $\Theta(N^2)$, avec N la taille du vecteur d'états. C'est pourquoi, ces méthodes sont le plus souvent utilisées dans des environnements de petite taille avec une gestion restrictive des amers visuels. Une autre approche probabiliste, nommée FastSLAM et publiée par Montermerlo *et al.* [54], consiste à remplacer le filtre de Kalman par un filtre à particules. La complexité algorithmique est alors de $\Theta(M \log(N))$, avec M le nombre de particules. Un ensemble de particules représente la densité de probabilité de la pose du robot. Chaque particule contient à la fois une hypothèse de trajectoire et une hypothèse de carte. Tandis que la trajectoire est estimée par le filtre à particules, chaque amer visuel contenu dans les cartes est estimé indépendamment par un filtre de Kalman de petite dimension. L'approche FastSLAM a ensuite été appliquée au SLAM visuel [55, 56].

2.2 SLAM par ajustement de faisceaux

L'ajustement de faisceaux (AF) est la technique consistant à optimiser un ensemble de paramètres, poses et amers visuels 3D, afin de minimiser l'erreur de reprojection (*cf.* section 1.3). L'algorithme classiquement utilisé est la méthode de moindres carrés non-linéaires de Levenberg-Marquardt qui est détaillée dans la section 1.5.2.

Les reconstructions à base d'AF sont en général plus précises que les méthodes probabilistes [57, 58, 59, 60] mais elles souffrent d'une complexité de calcul plus importante $\Theta(N^3)$, avec N le nombre de paramètres optimisés. On peut dès lors distinguer : les méthodes hors-ligne (très lente) utilisant un maximum de paramètres pour la reconstruction 3D et l'estimation de la trajectoire ; et les méthodes en-ligne, qui réduisent le nombre de paramètres à optimiser pour traiter le flux vidéo en temps réel [1]. Il est possible de réduire dans une certaine mesure les temps de traitement en améliorant le code informatique, en augmentant la puissance de calcul, ou en réduisant la complexité algorithmique du problème de reconstruction. C'est pourquoi, dans la majorité des approches, la réduction du temps de calcul passe par une réduction du nombre de paramètres à optimiser.

L'AF est particulièrement utilisé pour les méthodes hors-ligne. Parmi les logiciels de reconstruction 3D populaires, on peut citer Arc3D, Acute3D, Autodesk 123DCatch, Insight 3D, Bundler, PMVS2, FIT3D, VisualSFM, Hypr3D, APERO, 3DSOM, Photoscan, PhotoBuilder. Ces logiciels permettent de reconstruire un modèle 3D texturé de l'environnement à partir d'une liste d'images (voir les illustrations 2.1). Il faut toutefois compter sur des temps de calcul de l'ordre de plusieurs heures. Il est possible d'accélérer le temps de traitement des méthodes hors-ligne en subdivisant le problème de reconstruction en plusieurs problèmes de plus petites tailles. Cette approche a été utilisée par Royer *et al.* [61] avec structure hiérarchique : la séquence d'images est divisée récursivement en sous-séquences jusqu'à obtenir des triplets d'images. Le premier triplet est reconstruit avec l'algorithme de Nister présenté dans la section 3.1.1. Le triplet suivant est initialisé à partir des images communes et la $3^{\text{ème}}$ est calculée avec un calcul de pose (*cf.* 3.2.2). Les triplets sont fusionnés puis optimisés par AF. Le processus est répété jusqu'à parcourir l'intégralité de la structure hiérarchique. Dans le cadre du projet VIPA [2], cette méthode (hors-ligne) est utilisée pour reconstruire la trajectoire d'un véhicule. La reconstruction 3D est ensuite utilisée en temps réel pour localiser et guider un véhicule autonome le long de cette trajectoire (*cf.* figure 2.1e).

Contrairement aux méthodes hors-ligne où l'ensemble des images est connu à l'avance, les méthodes dites en-ligne traitent le flux vidéo image par image et en temps réel. L'idée est d'utiliser une approche incrémentale reconstruisant petit à petit l'environnement. Chaque étape de reconstruction n'implique qu'un sous-ensemble des paramètres considérés utiles, correspondant en général aux paramètres de la fin de la trajectoire (cartographie locale contenant les paramètres récemment utilisés) car potentiellement remis en cause par les observations faites dans les dernières images. Cette approche est utilisée dans toutes variantes de SLAM par AF, dont voici une liste non exhaustive [62, 63, 64, 65, 66, 67, 3, 4, 68, 5, 1, 2, 69].

Naturellement, il existe différentes manières d'interpréter la notion de « fin de trajectoire ». Plusieurs approches sont possibles pour sélectionner les paramètres à optimiser. Toutefois, celles-ci s'appuient sur la sélection proposée par Steedly *et al.* [62]. L'idée est de considérer que si une pose caméra est optimisée (*i.e.* les degrés de liberté relatifs à cette pose sont relâchés), alors les amers 3D observés par cette pose sont aussi optimisés. Puis les poses observant ces points 3D sont également relâchées. Steedly *et al.* proposent de répéter ce schéma, en partant de la pose caméra la plus récente, jusqu'à obtenir un système stable. En utilisant cette méthode de sélection, le nombre de paramètres peut s'avérer trop important. Pour viser des applications temps réel, Mouragnon *et al.* [67] ont proposé

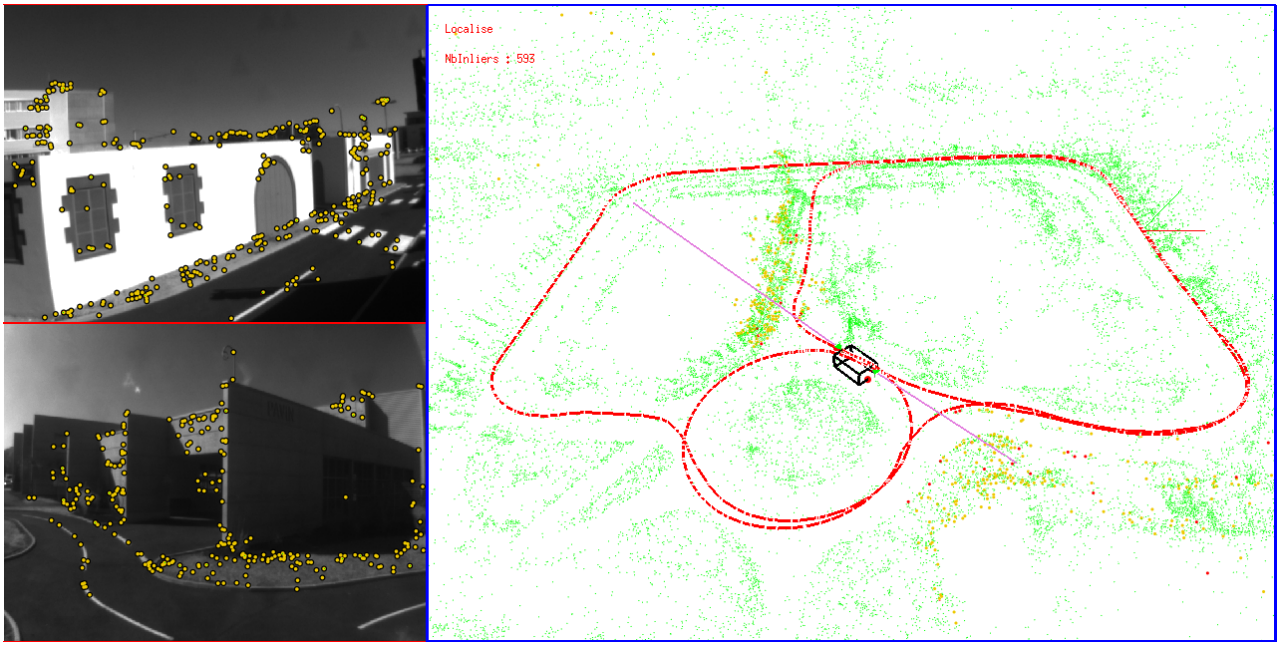
1. On parle de méthode temps réel si le temps de traitement d'une image est inférieur à la durée séparant deux images consécutives.
2. Véhicule autonome équipé d'une paire de caméras synchronisées à champs non recouvrant.

(a) Google Map

(b) Bundler

(c) VisualSFM

(d) Acute3D



(e) Algorithme VIPA. Les images à gauche correspondent aux caméras avant et arrière avec les points d'intérêt utilisés, la vue 3D correspond à une visualisation du modèle d'environnement, avec en vert les points 3D, et en rouge la trajectoire du véhicule.

FIGURE 2.1 – Méthodes de reconstruction 3D hors-ligne par ajustement de faisceaux.

(a) 6 *dof.* SLAM de Lim *et al.* [1] (b) Orb-SLAM de Mur-Artal *et al.* [2]

FIGURE 2.2 – Algorithmes de SLAM en environnement urbain sur la base KITTI [70].

(a) PTAM de Klein et Murray [3] (b) SLAM de Tamaazousti *et al.*[4] (c) SLAM de Larnaout *et al.* [5]

FIGURE 2.3 – Algorithmes de SLAM appliqués à la Réalité Augmentée.

de limiter les paramètres optimisés aux 3 dernières poses, et aux points 3D vus dans les 10 dernières poses. Klein et Murray [3] se limitent quant à eux aux 5 poses les plus rapprochées spatialement de la pose courante.

La méthode employée dans nos travaux, et par Mur Artal *et al.* [2], est la méthode de Steedly *et al.* répétée deux fois (*cf.* la sous-section 3.3.3 pour plus de détails). Cela permet un compromis entre sélection des données les plus pertinentes et temps de calcul. Le point important dans les approches d'AF incrémental est d'inclure dans la fonction de coût les erreurs liées aux amers 3D vus dans des poses caméras non optimisées. Ceci a pour effet d'ancrer la partie plus ancienne de la trajectoire et limite la dérive induite par l'accumulation d'erreurs.

2.3 Conclusion

Les applications visées dans ce travail de thèse concernent la robotique mobile et la Réalité Augmentée. Ainsi, on souhaite privilégier la qualité de la localisation par rapport à la précision de reconstruction de la carte. Or, plusieurs travaux [57, 58, 59, 60] montrent que l'AF est plus précis que les approches probabilistes. En contrepartie, celui-ci peut nécessiter plus de calculs. Toutefois, les implémentations à base de matrices éparses ainsi que les progrès faits en matière d'architecture matérielle et de compilation permettent désormais l'utilisation d'AF en temps réel même sur des architectures mobiles [71]. C'est pourquoi, le travail présenté sera focalisé sur les approches par AF, avec notamment un SLAM par images clés présenté dans le chapitre suivant.

Chapitre 3

Algorithme de SLAM Visuel

Ce chapitre présente les différentes étapes constituant l'algorithme de SLAM visuel développé au cours de ce travail de thèse et illustré par la figure 3.1. Après une étape d'initialisation de la carte (3.1), le processus de localisation (3.2) extrait les points d'intérêt de chaque image du flux vidéo afin d'effectuer une mise en correspondance (3.2.1) avec le modèle 3D de l'environnement. Puis ces associations 2D-3D sont utilisées pour calculer la pose de la caméra (3.2.2) au moment de la prise de l'image (on parlera également de « poses des images »). Lorsqu'il est nécessaire de mettre à jour le modèle d'environnement (3.3), une image clé est ajoutée à la carte, et celle-ci est raffinée par AF afin de conserver un modèle précis et à jour. Un intérêt tout particulier a donc été porté sur la précision et la rapidité d'exécution des méthodes utilisées. De plus, l'approche se veut générique sur les méthodes de détection et de description des points d'intérêt. Dans cette partie, le détecteur utilisé est celui de Harris et Stephen et le descripteur est la ZNCC (*cf.* section 1.6.1). Les étapes de l'algorithme sont résumées dans le diagramme fonctionnel 3.1 et détaillées dans ce chapitre. Pour chacune de ces étapes, différentes alternatives présentes dans l'état de l'art sont également présentées et discutées.

3.1 Initialisation

Cette section présente 2 méthodes pour initialiser la trajectoire et la carte du SLAM. Tout d'abord, un triplet d'images est sélectionné afin que le déplacement entre chaque image permette l'utilisation de la géométrie épipolaire et la triangulation des amers 3D. L'heuristique proposée ici sélectionne un triplet d'images $I = \{I_1, I_2, I_3\}$ tel que 75% des amers détectés dans I_1 soient appariés avec les amers détectés dans I_2 et 50% soient appariés avec les amers détectés dans I_3. Pour augmenter la fiabilité des appariements, Mouragnon [67] propose de suivre les amers détectés de proche en proche sur chacune des images intermédiaires. Deux méthodes sont présentées pour estimer la trajectoire et les points à partir de trois images $I = \{I_1, I_2, I_3\}$ dans le cas où la caméra est calibrée :

1. l'algorithme des 5 points dans 3 vues de Nister *et al.* [72],
2. l'algorithme des 5 points dans 2 vues de Lui et Drummond [73] pour estimer l'orientation combiné à une recherche stochastique pour estimer la translation.

En l'absence d'*a priori* sur la position initiale de la caméra, le repère monde est fixé arbitrairement.

3.1.1 Nister

L'algorithme des 5 points dans 3 vues présenté par Nister [72] est la méthode de référence pour le calcul de poses relatives. Cette méthode est basée sur la méthode des 5 points dans 2 vues présentée par Nistér dans [74] où un processus RANSAC (*cf.* 1.10) estime la matrice essentielle à partir de 5 points observés dans I_1 et I_3. La méthode des 5 points dans 2 vues résout le calcul de matrice essentielle avec un système d'équations polynomiales de degré 10. Pour le détail des calculs, le lecteur

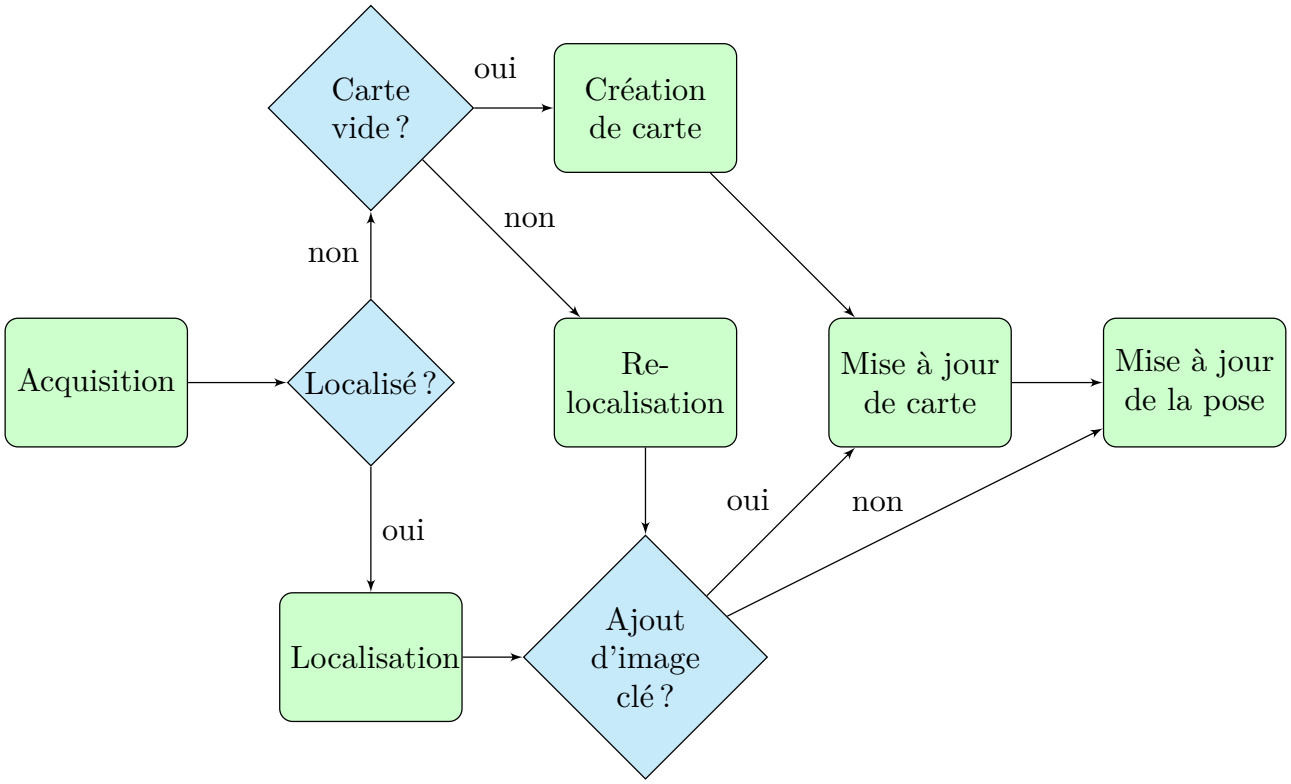


FIGURE 3.1 – Diagramme fonctionnel du SLAM allant de l'acquisition à la mise à jour de la pose en passant par la localisation et la mise à jour de la carte. Ce schéma prend en compte l'initialisation de la carte ainsi que l'appel à une méthode de « re-localisation » lorsque la localisation normale échoue.

intéressé pourra se référer aux travaux de Royer [31]. On obtient alors le déplacement relatif de la caméra entre I_1 et I_3. En fixant arbitrairement la position de I_1, on déduit la position relative de I_3 afin de trianguler les 5 points 3D. Puis on calcule la pose I_2 à partir des 5 mises en correspondance observées dans l'image I_2. Le triplet de poses, ainsi que les points 3D associés, sont ensuite raffinés dans un ajustement de faisceaux global optimisant à la fois les paramètres de poses et de points 3D. La carte obtenue est utilisée comme carte de référence pour l'étape de localisation.

L'inconvénient de cette méthode provient du calcul de la matrice essentielle qui génère 4 solutions. Un post-traitement est nécessaire pour déterminer la solution qui minimise la somme des erreurs de reprojection. Bien que la méthode de Nister soit la référence dans le domaine, le RANSAC ainsi que la recherche de la meilleure solution peuvent engendrer un temps de calcul relativement long. De plus, un système de polynômes avec un degré élevé peut être mal conditionné et difficile à implémenter [75].

3.1.2 Méthode de Lui et Drummond

La méthode de Lui et Drummond [73] utilise une résolution itérative pour calculer la matrice essentielle à partir de 5 appariements afin d'obtenir l'orientation relative entre deux poses. La méthode est illustrée par la figure 3.2. Les poses sont représentées par des sphères unitaires de centre C et C'. L'objectif est de calculer l'orientation des 3 poses. Pour cela, on définit pour chaque appariement le plan épipolaire (en bleu sur la figure) défini par les droites épipolaires et les centres C et C'. Pour chaque appariement, la méthode minimise les angles entre le plan épipolaire et les projetés des droites épipolaires sur les sphères voisines $\hat{V}$ et $\hat{V}'$. L'avantage de cette méthode est d'être robuste et rapide à calculer. De plus, les résolutions itératives sont en général plus simples et rapides comparativement aux systèmes de polynômes comme l'algorithme de Nister.

Toutefois, la méthode de Lui et Drummond ne permet pas de calculer la translation entre les deux poses. Ainsi il est nécessaire d'avoir recours à une seconde passe, à la fois pour affiner l'orientation obtenue mais également pour déterminer la translation. Pour cela, on génère empiriquement n hypothèses de déplacement uniformément réparti (en pratique 27 déplacements sont générés). Chacune des hypothèses est évaluée à travers un processus d'optimisation minimisant la somme des distances épipolaires et des erreurs de reprojection :

$$\mathcal{E} = \underset{C',P}{\arg\texttt{Min}} \sum_{i}^{N} \rho(\varpi(C, p_{1,i}, C', p_{2,i}), c_1) + \varphi \sum_{j}^{M} \rho(\pi(C, P_j) - p_{k,j} + \pi(C', P_j) - p_{k,j}, c_2) \tag{3.1}$$

avec N le nombre d'appariements utilisés, M le nombre de points 3D triangulés, P_j le j$^{\text{ième}}$ point 3D, et $p_{k,i}$ le i$^{\text{ième}}$ amer 2D dans l'image k. Les fonctions de distance épipolaire et de projection sont notées respectivement ϖ et π. Les mauvais appariements sont rejetés en utilisant le M-Estimateur Geman McClure noté ρ dont les coefficients c_1 et c_2 sont calculés à partir de la médiane des résidus (pour plus de détails, *cf.* section 3.2.2). Une première minimisation de $\mathcal{E}$ est effectuée avec $\varphi = 0$ pour optimiser les 6 degrés de liberté de la seconde pose en utilisant la contrainte épipolaire. Sachant que l'on recherche le déplacement relatif, la première pose est considérée fixe. Puis les points 3D P sont triangulés et une seconde minimisation est effectuée avec $\varphi = 1$. En plus de la seconde pose, cette nouvelle étape optimise également les points 3D en minimisant l'erreur de reprojection dans les deux poses. Il est démontré que l'utilisation des erreurs de reprojection en complément de l'erreur épipolaire apporte une convergence plus rapide ainsi qu'une meilleure précision pour les méthodes d'AF [59].

Parmi les différentes hypothèses testées, on conserve celle qui obtient un maximum de points satisfaisants tels que le critère d'erreur de reprojection : $\|\pi(C_k, P_j) - p_{k,j}\| < 2.0$. L'évaluation de chaque hypothèse étant indépendante, il est possible de paralléliser le traitement de manière triviale avec la bibliothèque OpenMP [76].

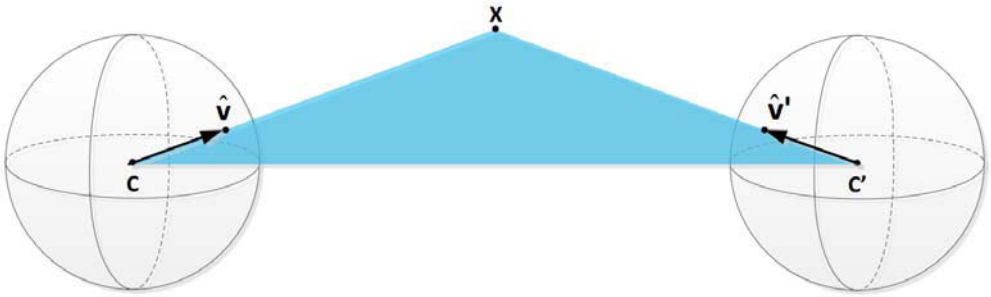


FIGURE 3.2 – Méthode de Lui et Drummond pour l'estimation de l'orientation qui consiste à minimiser l'angle entre le plan épipolaire en bleu et les projetés des épipoles sur les sphères. Crédit image : Lui et Drummond [73]

3.2 Localisation

La localisation estime les 6 paramètres de pose de la caméra au moment de l'acquisition de l'image en utilisant la carte 3D déjà reconstruite comme modèle de référence. Une représentation minimale de la pose nécessite 3 paramètres pour la translation et 3 paramètres pour l'orientation avec la représentation exponentielle locale (pour plus de détails, voir la section 1.4). La carte est constituée d'un ensemble d'amers 3D ainsi que des descripteurs associés. En pratique, il n'est pas nécessaire d'utiliser toute la carte 3D pour localiser la pose courante. Une solution consiste à utiliser la dernière pose connue de la caméra pour sélectionner les amers 3D potentiellement présents dans le champ de vue. Il est également possible de prédire la pose courante de la caméra en utilisant un modèle de prédiction. Toutefois, la solution que l'on utilise ici consiste à sélectionner un historique d'amers 3D proches, spatialement et temporellement, défini par l'ensemble des amers 3D optimisés lors du dernier processus de mise à jour de la carte (*cf.* la section sur la sélection des paramètres à optimiser 3.3.3).

La localisation se divise en 2 étapes : une mise en correspondance entre l'image courante et les points 3D déjà reconstruits (constituant le modèle de référence) afin d'obtenir une liste d'appariements, puis un calcul de pose à partir des appariements obtenus.

3.2.1 Mise en correspondance

La mise en correspondance consiste, dans un premier temps, à détecter des amers visuels dans l'image courante pour, dans un second temps, effectuer un appariement avec les amers contenus dans le modèle de référence. Afin de limiter le temps de calcul, on suppose le déplacement entre deux poses successives faible. Puis, on prédit les zones où l'on est censé retrouver les amers connus. Pour cela, on projette dans l'image courante les amers 3D connus en utilisant la dernière pose connue comme approximation de la pose courante. Ainsi, chaque amer projeté n'est apparié qu'avec des détections proches dans l'image. La dimension de la zone de recherche est déterminée empiriquement en fonction des mouvements et de la fréquence de la caméra (en pratique, on utilise une fenêtre de 20×20 pixels pour une caméra portée à la main cadencée à 60 Hz). Cette étape est illustrée par la figure 3.3, avec les zones de recherche (rectangles jaunes) qui sont centrées sur les amers projetés (points verts) et les appariements avec les détections (points rouges) qui sont représentés par des traits bleus. Il résulte de cette étape une liste d'associations 2D-3D où chaque amer 2D apparié dans l'image courante est associé à un amer 3D issu de l'image clé.

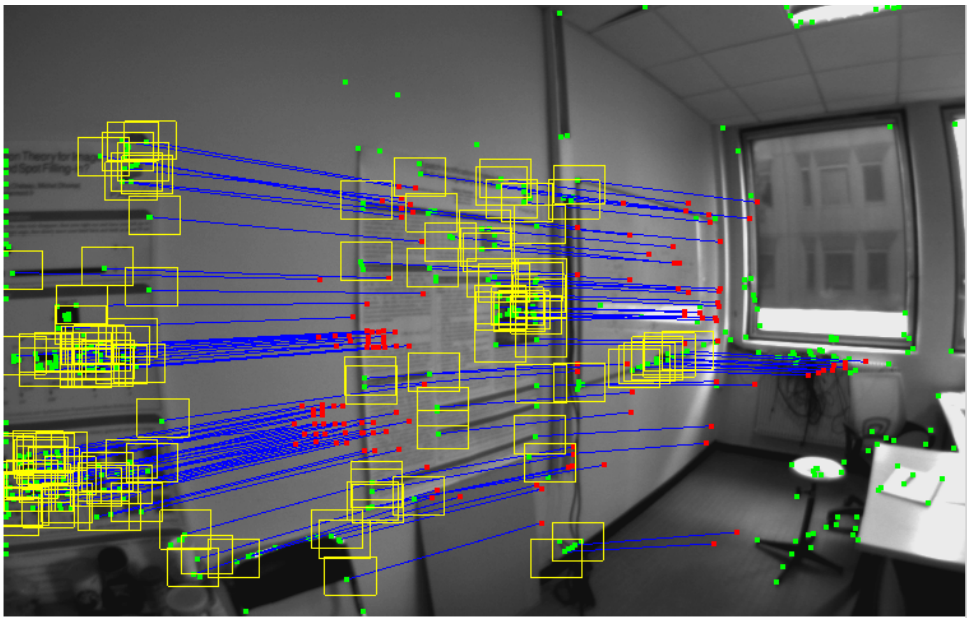


FIGURE 3.3 – Appariement prédictif : on observe en vert les points d'intérêt de l'image courante appariés avec les points d'intérêt de l'image clé ici virtuellement représentée en rouge. Les rectangles jaunes correspondent aux zones de prédiction dans lesquelles sont sélectionnés les points à apparier.

3.2.2 Calcul de pose

Dans cette étape, on estime la pose courante de la caméra en minimisant la somme des erreurs de reprojection des associations 2D-3D (issues de la phase d'appariement 3.2.1). En l'absence de faux appariements, le problème se résumerait à une simple optimisation avec une méthode de moindres carrés non linéaires. En pratique, il est nécessaire d'utiliser des méthodes robustes pour prendre en compte les mauvaises associations. Différentes méthodes robustes de l'état de l'art sont présentées ci-dessous. Dans cette section, on appelle *inlier* une association 2D-3D dont l'erreur de reprojection est inférieure ou égale à 2 pixels, et *outlier* lorsque l'erreur est supérieure à 2 pixels. Quelle que soit la méthode utilisée, on considère que la localisation a réussi si la pose calculée permet d'obtenir plus de 30 *inliers*.

RANSAC

L'algorithme de RANSAC a été développé pour les problèmes de calcul de pose (*cf.* section 1.10). A chaque itération de RANSAC, une pose est calculée à partir de 3 points (choisis aléatoirement) en utilisant l'algorithme P3P proposé par Kneip *et al.* [77]. A la fin des itérations, on conserve la pose C qui minimise la fonction Λ correspondant au nombre d'*outliers*, avec P_i le point 3D associé au point d'intérêt p_i :

$$\Lambda = \sum_{i=0}^{N} \rho(\pi(C, P_i) - p_i) \tag{3.2}$$

avec

$$\rho(\varepsilon) = \begin{cases} 0 & \text{si } \|\varepsilon\| \leq 2 \\ 1, & \text{sinon} \end{cases}$$

La meilleure solution obtenue est ensuite raffinée en utilisant l'algorithme de Levenberg-Marquardt avec uniquement les associations 2D-3D *inliers* pour minimiser la fonction de coût suivante :

$$\mathcal{E} = \arg\underset{C}{\texttt{Min}} \sum_{i=0}^{N} \rho_1(\pi(C, P_i) - p_i) \tag{3.3}$$

avec

$$\rho_1(\varepsilon) = \begin{cases} \varepsilon^2 & \text{si } \|\varepsilon\| \leq 2 \\ 0, & \text{sinon} \end{cases}$$

Médiane

La méthode robuste dite « Médiane » utilise la médiane des erreurs de reprojection notée *Med*. Cette méthode fait l'hypothèse que la moitié des associations 2D-3D, au minimum, est correcte. Ainsi, la médiane des erreurs est mise à jour à chaque itération et une association 2D-3D est considérée *inlier* si son erreur est inférieure à $max(Med(r), 2)$ avec ϵ le vecteur de résidus. Les paramètres de la pose C sont initialisés avec la dernière pose connue puis optimisés par l'algorithme de Levenberg-Marquardt (voir section 1.5.2). La méthode minimise les erreurs de reprojection des associations satisfaisant le critère médian ϕ :

$$\mathcal{E} = \arg\underset{C}{\texttt{Min}} \sum_{i=0}^{N} \phi(\pi(C, P_i) - p_i, Med) \tag{3.4}$$

avec

$$\phi(\varepsilon, Med) = \begin{cases} \varepsilon^2 & \text{si } \|\varepsilon\| \leq max(Med, 2) \\ 0, & \text{sinon} \end{cases}$$

Toutefois, cette méthode nécessite un comptage *a posteriori* des *inliers* car la pose peut avoir été calculée avec une erreur médiane élevée notamment lorsque la liste d'associations 2D-3D contient trop d'erreurs. Le comptage des *inliers* permet alors de vérifier si le résultat est exploitable.

M-Estimateur

Une méthode plus simple à mettre en œuvre consiste à utiliser un M-Estimateur (pour plus de détails sur les M-Estimateurs, voir [78]). Cela permet une gestion implicite des *outliers* : d'une part, toutes les associations sont utilisées dans le calcul et d'autre part, le M-Estimateur marginalise dans la fonction de coût les associations 2D-3D ayant une erreur trop importante. La méthode utilisée ici est un moindres carrés pondérés itératifs IRLS (Iteratively Reweighted Least Squares) utilisant le M-Estimateur de Geman-McClure. Les méthodes IRLS ont l'avantage d'être faciles à implémenter car elles ne modifient pas la jacobienne de la fonction de coût : la pondération s'applique sur les équations normales en multipliant la jacobienne par une matrice de pondération qui découle du M-Estimateur. La fonction de coût à minimiser pour l'algorithme IRLS est la suivante :

$$\mathcal{E} = \arg\underset{C}{\texttt{Min}} \sum_{i=0}^{N} \rho(\pi(C, P_i) - p_i, c) \tag{3.5}$$

avec ρ le M-Estimateur de Geman-McClure défini par :

$$\rho(\varepsilon, c) = \frac{\varepsilon^2}{\varepsilon^2 + c^2} \tag{3.6}$$

et c étant la valeur du résidu au-delà de laquelle l'observation est considérée aberrante. Ici on utilise la valeur proposée par Lothe *et al.* [79] :

$$c = \text{mediane}(\epsilon) + 5.2 \times \text{MAD}(r) \tag{3.7}$$

avec MAD pour *Median Absolute Deviation* défini par :

$$\text{MAD}(\epsilon) = \text{mediane}(\mid \epsilon - \text{mediane}(\epsilon) \mid) \tag{3.8}$$

3.3 Mise à jour de carte

L'étape de reconstruction incrémentale consiste à mettre à jour la carte 3D à partir des données observées au fur et à mesure que la caméra se déplace. Une première étape consiste à déterminer si l'image courante doit être sélectionnée comme une image clé (3.3.1) et ajoutée à la carte de référence. Lorsque c'est le cas, la carte doit être actualisée afin de tenir compte des nouvelles observations et donc de nouveaux points 3D. Ceux-ci sont obtenus par appariements et triangulation (3.3.2). L'ajout de nouvelles informations peut remettre en cause les données déjà présentes dans la carte. C'est pourquoi, les paramètres de la carte doivent être ré-optimisés à chaque ajout d'image clé (3.3.3). Toutefois, la quantité d'information contenue dans la carte étant potentiellement importante, les méthodes de SLAM par AF ont recours à une optimisation locale sur un sous-ensemble de paramètres de la carte.

3.3.1 Sélection d'images clés

Le processus de sélection d'images clés consiste à déterminer s'il est pertinent d'ajouter l'image courante dans la carte afin de mettre à jour et d'enrichir celle-ci. Ce critère de sélection est un compromis minimisant le nombre d'images clés et assurant un recouvrement du champ visuel suffisant entre deux images clés successives. Il existe certainement autant de critères de sélection d'images clés que d'algorithmes de SLAM basés AF. Quatre critères sont présentés ici.

Critère de Mouragnon *et al.* [67]

Dans l'approche de Mouragnon *et al.*, une image est sélectionnée comme image clé si les conditions suivantes sont réunies :

- le nombre d'appariements avec l'image clé précédente est supérieur à 400,
- l'incertitude sur la pose est inférieure à un seuil,
- le nombre de points *inliers* est supérieur à 10.

Le nombre d'appariements ne constitue pas un critère fiable car il ne prend pas en compte les mauvais appariements. L'incertitude sur la pose est un bon critère, toutefois, le calcul de celle-ci nécessite d'être approximé pour ne pas dégrader les performances de temps de calcul.

Critère de Klein *et al.* [3]

Le critère de sélection d'images clés défini par Klein *et al.* est :

- un nombre d'*inliers* minimum,
- au moins 20 images ont été acquises depuis la dernière image clé,
- une distance 3D minimale entre la pose courante et la dernière image clé.

Les conditions imposant un certain nombre d'images ainsi qu'un déplacement entre deux images clés assurent un parallaxe suffisant pour la triangulation des points et empêchent la prise d'image lorsque la caméra est statique. Toutefois, le nombre d'images entre chaque image clé dépend du capteur et donc de l'application, tandis que la distance minimale 3D est difficile à définir dans un cadre de SLAM monoculaire où le facteur d'échelle est inconnu.

Critère de Mur Artal *et al.* [2]

Le critère de sélection d'images clés défini par Mur Artal *et al.* est :
— au moins 50 *inliers*
— au moins 10 images ont été acquises depuis la dernière image clé
— moins de 90% des points 3D optimisés lors du dernier AF incrémental ont été retrouvés.

Contrairement à Klein *et al.*, Mur Artal *et al.* utilisent un ratio sur les points 3D retrouvés lors du calcul de pose. Ce critère a l'avantage de ne pas utiliser de distance 3D. Toutefois, certaines configurations peuvent poser des problèmes : si 60 points 3D sont présents dans la base, et que la localisation en retrouve 55, plus de 90% des points 3D ont été vus donc le critère ne sélectionne pas de nouvelle image clé. Si à l'image suivante, seulement 49 points 3D sont retrouvés, le critère ne satisfait pas la condition sur les 50 *inliers* et l'image n'est toujours pas sélectionnée comme image clé. Étant donné que la carte n'a pas été mise à jour, la localisation échouera probablement à l'image suivante.

Proposition de critères

Idéalement, le critère utilisé doit sélectionner une nouvelle image clé si et seulement si celle-ci contient de nouvelles informations exploitables. Existe-t-il un critère qui soit robuste aux configurations théoriques suivantes ?
— la caméra est statique et de nouveaux objets statiques apparaissent petit à petit dans l'environnement jusqu'à ce que celui-ci soit totalement renouvelé : la caméra étant immobile, le critère de Klein *et al.* empêcherait la prise d'une nouvelle image clé jusqu'à ce que la localisation échoue,
— la caméra est statique et les objets de l'environnement sont petit à petit remplacés par du blanc uniforme : les critères basés sur le nombre d'appariements ou le nombre d'*inliers* déclencheront la prise d'une nouvelle image clé alors qu'aucune nouvelle information n'est venue enrichir l'environnement.

Dans notre implémentation, l'image clé sélectionnée est la dernière image ayant satisfait le critère suivant :

$$\frac{\phi(n_{\text{inliers}})}{n_{\text{détections}}} > 0.2 \tag{3.9}$$

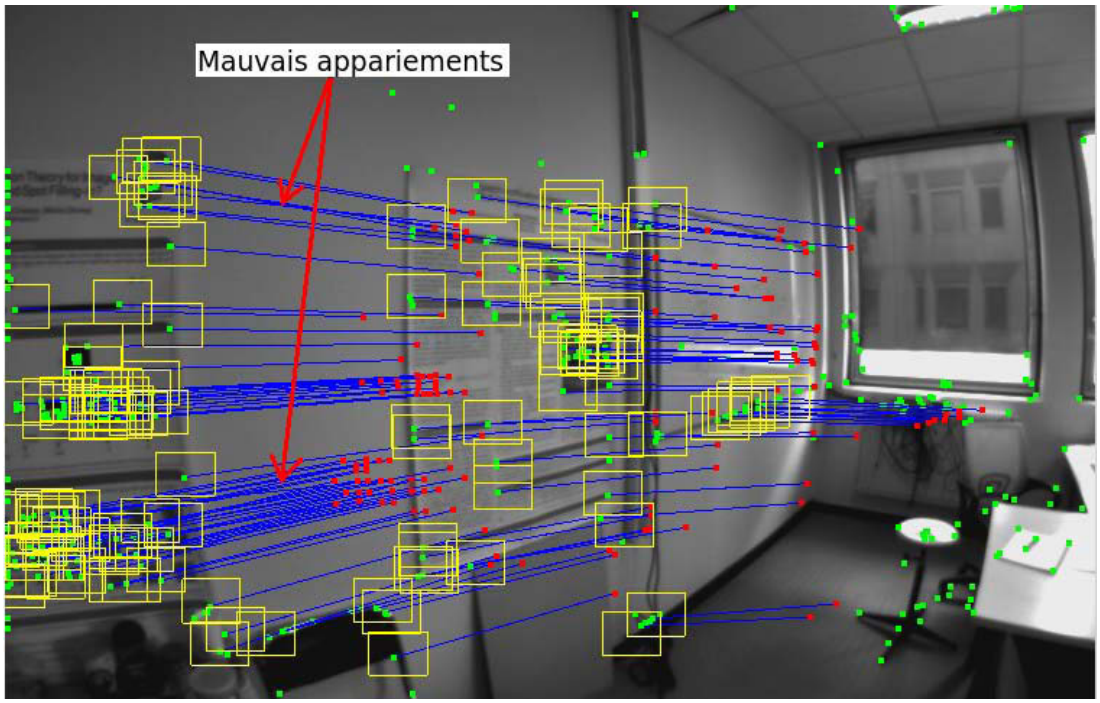
où

$$\phi(n_{\text{inliers}}) = \begin{cases} n_{\text{inliers}} & \text{si } n_{\text{inliers}} \geq 50 \\ 0, & \text{sinon} \end{cases}$$

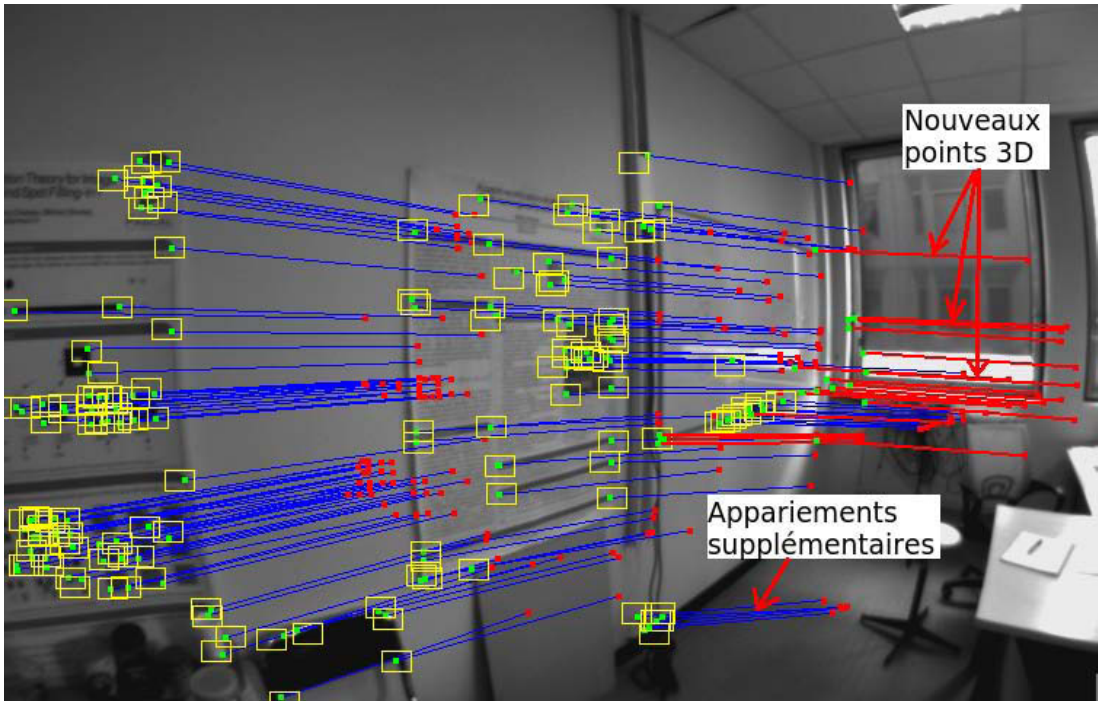
avec n_{inliers} le nombre d'inliers et $n_{\text{détections}}$ le nombre d'amers détectés dans l'image. Ce critère requiert que l'image sélectionnée contienne au minimum 50 *inliers* et que 20% des détections correspondent à des amers 3D déjà présents dans la carte. Utiliser un ratio en fonction du nombre de points détectés permet de ne pas sélectionner d'images clés lorsque l'environnement s'appauvrit en information, car le nombre de points détectés diminuera également. De plus, lors d'un mouvement brusque, l'image sélectionnée sera l'image qui précède l'image floue donc la dernière image correctement localisée.

3.3.2 Appariement et triangulation

Cette sous-section présente l'enrichissement de la carte 3D qui se compose d'une étape d'appariement entre l'image courante et la carte et d'une étape de triangulation des nouveaux points 3D. Cette dernière utilise les détections courantes appariées avec les points d'intérêt déjà présents dans la base et qui ne sont pas associés à un point 3D.



(a) Rappel de la figure 3.3 : appariements effectués pour la localisation. Quelques mauvais appariements sont pointés. On observe dans la figure 3.4b que l'utilisation de la pose calculée pour améliorer la prédiction des points d'intérêt permet de supprimer ces mauvais appariements.



(b) Nouveaux appariements utilisant la pose courante pour prédire plus efficacement la position des points d'intérêt. On observe que les mauvais appariements de l'image 3.4a ont disparu. Les points d'intérêt verts de l'image courante sont appariés avec les points d'intérêt de l'image clé virtuellement représentés par les points rouges. Les rectangles jaunes correspondent aux zones de prédiction dans lesquelles sont sélectionnés les points à apparier. Les appariements qui vont permettre la création de nouveaux points 3D sont représentés par les segments rouges.

FIGURE 3.4 – Comparatif entre l'appariement effectué à l'étape de localisation et le nouvel appariement pour la mise à jour de la carte.

Appariement

Afin d'être intégrées à la base, les nouvelles observations sont associées aux anciennes données de la carte 3D. Les appariements entre l'image courante et la base sont calculés lors de la localisation (3.2.1). Toutefois, il est préférable d'effectuer un nouvel appariement en utilisant la pose calculée lors de la localisation pour prédire plus précisément la position des points dans l'image (par projection des points 3D) et ainsi réduire le nombre de faux appariements. Cela est réalisé en réduisant la dimension des zones de recherche des appariements, en pratique elles sont divisées par 2. Le nombre de faux appariements est alors moins important que lors de la localisation. Il est également possible d'avoir des appariements supplémentaires comme le montre la figure 3.4. Le résultat est une meilleure propagation de l'information de la carte avec des points 3D observés dans de nombreuses vues.

Il n'est pas possible de prédire la position des points d'intérêt non associés à des points 3D. C'est pourquoi ces points sont appariés avec les candidats proches de la droite épipolaire (*cf.* 1.8). Une illustration est faite figure 3.4b, avec les points d'intérêt en rouge issus de l'image clé de référence, les points d'intérêt en vert issus de l'image clé courante, les zones de recherche des points d'intérêt associés à un point 3D en jaune, les lignes bleues pour les appariements avec les points prédits et les lignes rouges pour les appariements avec les points appariés avec la contrainte épipolaire. Cette étape de mise en correspondance combine deux stratégies d'appariement (par zone et par distance épipolaire) exploitant la pose précise issue de la localisation afin d'assurer un enrichissement et une propagation fiable des données de la carte. Les points qui ont été appariés avec la contrainte épipolaire ne sont pas encore associés à un point 3D. Pour initialiser un point 3D à partir de deux observations, on a recours à la triangulation présentée dans la sous-section suivante.

Triangulation

Le processus permettant la meilleure estimation d'une position de point 3D est la minimisation de l'erreur de reprojection en utilisant l'ensemble de ses observations 2D. Toutefois, ce processus nécessite une initialisation du point 3D que l'on peut obtenir par triangulation. En utilisant une caméra calibrée, la triangulation d'un point 3D requiert deux observations à partir de deux points de vues différents (avec une seule observation, la profondeur ne peut être déterminée). La triangulation consiste à calculer l'intersection des rayons optiques issus des centres optiques des poses caméras et passant par les observations. En pratique, ces rayons peuvent ne pas s'intersecter. Ceci est dû aux bruits sur les observations et aux approximations de modèle (paramètres intrinsèques et extrinsèques des caméras). On utilise communément la méthode du point du milieu [80] détaillée dans la section 1.9. Bien que la méthode peut être étendue à plusieurs poses et observations en calculant le barycentre des différents points du milieu obtenus, on préfère utiliser la minimisation de l'erreur de reprojection pour mettre à jour les points 3D avec les nouvelles observations. Une alternative proposée par Hartley *et al.* [81] est de trianguler un point observé dans plusieurs vues en utilisant la méthode *Direct Linear Transform.*Le lecteur intéressé pourra se référer aux travaux de Hartley *et al.* [82] pour une comparaison des différentes méthodes de triangulation.

3.3.3 Optimisation incrémentale

L'optimisation incrémentale présentée ci-dessous s'effectue en deux étapes. La première consiste à sélectionner un sous-ensembles de paramètres constitué à la fois d'anciennes et de nouvelles données afin d'intégrer la nouvelle image clé et les nouveaux points 3D de façon cohérente dans la base. La seconde étape optimise les paramètres sélectionnés afin de minimiser la somme des erreurs de reprojection du problème. La méthode utilisée est un ajustement de faisceaux (AF).

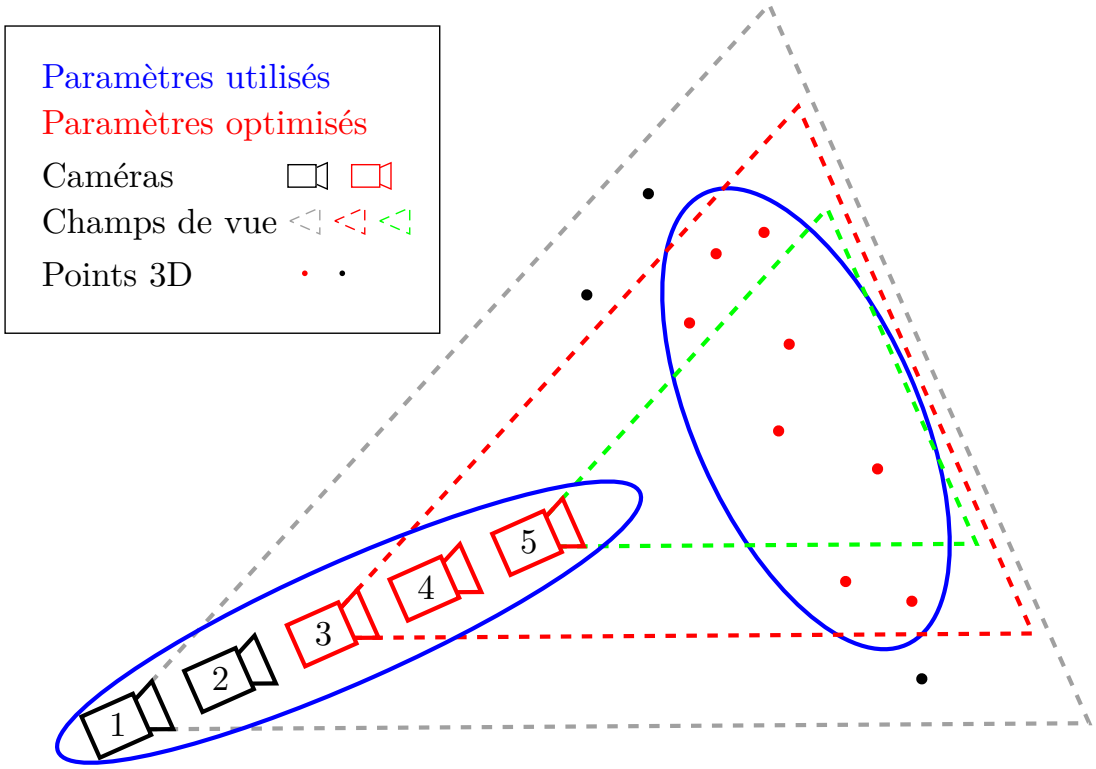


FIGURE 3.5 – Sélection des paramètres et observations pour l'ajustement de faisceaux incrémental.

Sélection des paramètres à optimiser

La sélection des paramètres (poses caméra et points 3D) intervenant dans la fonction de coût ainsi que les paramètres à optimiser se fait selon le schéma décrit par Steedy *et al.* [62] à l'ordre 1. Cette procédure est illustrée par la figure 3.5 :

- sélection de l'ensemble P^{Pose_5} des points 3D observés par la dernière image clé (numérotée 5 et notée Pose_5),
- sélection de l'ensemble $\mathcal{C}$ des poses à optimiser et défini par les poses observant au moins un des points de P^{Pose_5}, donc $\mathcal{C} = \{\text{Pose}_3, \text{Pose}_4, \text{Pose}_5\}$,
- sélection de l'ensemble $\mathcal{P}$ des points 3D à optimiser et défini par l'ensemble des points 3D observés par les poses $\mathcal{C}$,
- les projections des points 3D de $\mathcal{P}$ dans l'ensemble des poses caméra où ils sont observés, donc $\{\text{Pose}_1, \ldots, \text{Pose}_5\}$, sont utilisés dans la fonction de coût (ce qui inclut des poses n'étant pas optimisées).

Puis un AF minimise la fonction de coût $\mathcal{E}$ avec l'algorithme de Levenberg-Marquardt en utilisant la méthode robuste IRLS avec le M-Estimateur de Geman McClure noté ρ (*cf.* 3.2.2) pour estimer l'ensemble de paramètres $\{\mathcal{C}, \mathcal{P}\}$ minimisant la somme de toutes les erreurs de reprojection de $\mathcal{P}$:

$$\mathcal{E} = \underset{\mathcal{C},\mathcal{P}}{\arg \texttt{Min}} \left(\sum_{i=0}^{I} \rho(\pi(C_{p_i}, P_{p_i}) - p_i) + \sum_{k=0}^{K} \rho(\pi(\bar{\mathcal{C}}_{p_k}, P_{p_k}) - p_k) \right) \tag{3.10}$$

avec I le nombre d'observations liées aux poses de $\mathcal{C}$ et aux points 3D de $\mathcal{P}$, et K le nombre d'observations liées aux poses non optimisées $\bar{\mathcal{C}}$. On note $\mathcal{C}_{p_i}$ la pose $C \in \mathcal{C}$ telle que l'observation p_i soit détectée dans l'image de la pose $C = \mathcal{C}_{p_i}$ et $P = \mathcal{P}_{p_i}$ l'amer 3D de $\mathcal{P}$ tel que π_{p_i} soit la projection de P dans C : $\pi_{p_i} = \pi(\mathcal{C}_{p_i}, \mathcal{P}_{p_i}) = \pi(C, P)$.

Ajustement de faisceaux

L'ajustement de faisceaux est une méthode héritée des travaux de recherche en photogrammétrie [83]. Désormais, l'AF est couramment utilisé en vision par ordinateur [81, 59]. Cette méthode minimise une fonction de coût correspondant aux erreurs de reprojection en utilisant l'algorithme des moindres carrés non linéaires Levenberg-Marquardt présenté dans la section 1.5.2. Avec J la jacobienne de la fonction d'erreur (donnée par l'équation 3.10) et ϵ les résidus (vecteur d'erreurs), chaque itération de Levenberg-Marquardt consiste à résoudre les équations normales du système suivant :

$$(J^T J + \lambda I)\Delta = J^T \epsilon \tag{3.11}$$

La complexité algorithmique de la résolution de ce système est en $\Theta(N^3)$ avec N le nombre de paramètres. Il est toutefois possible de réduire cette complexité en utilisant le complément de Schur.

Complément de Schur Le complément de Schur, aussi connu comme « l'astuce du complément de Schur », permet de réduire le problème d'inversion d'une matrice de dimension $(N+M)\times(N+M)$ en un problème d'inversion de deux matrices de taille respective $N\times N$ et $M\times M$, sous réserve que celles-ci soient inversibles. Pour cela, on ordonne les paramètres de la jacobienne tels que $J = \begin{bmatrix} J_C & J_P \end{bmatrix}$ avec J_C la jacobienne des poses et J_P la jacobienne des points 3D. Le système d'équations 3.11 prend la forme :

$$\begin{bmatrix} U & W \\ W^T & V \end{bmatrix} \begin{bmatrix} \Delta_C \\ \Delta_P \end{bmatrix} = \begin{bmatrix} \epsilon_C \\ \epsilon_P \end{bmatrix} \tag{3.12}$$

avec

$$U = J_C^T J_C + \lambda I, \quad V = J_P^T J_P + \lambda I, \quad W = J_C^T J_P, \quad \epsilon_X = J_X^T \epsilon \tag{3.13}$$

D'après la fonction de coût 3.10, les paramètres des points 3D sont indépendants entre eux, ce qui implique que les dérivées croisées entre deux points 3D soient nulles. Ainsi, la matrice V sera toujours une matrice symétrique diagonale par blocs (chaque bloc étant de dimension 3×3) donc triviale à inverser (en inversant chaque bloc indépendamment). Comme V est facile à inverser, on utilise le complément de Schur pour ré-écrire le système 3.12 en multipliant à gauche par $\begin{bmatrix} I & -WV^{-1} \\ 0 & I \end{bmatrix}$ ce qui donne :

$$\begin{bmatrix} U - WV^{-1}W^T & 0 \\ W^T & V \end{bmatrix} \begin{bmatrix} \Delta_C \\ \Delta_P \end{bmatrix} = \begin{bmatrix} \epsilon_C - WV^{-1}\epsilon_P \\ \epsilon_P \end{bmatrix} \tag{3.14}$$

Sous cette nouvelle forme, le système à résoudre se compose de deux équations :

$$S\Delta_C = \epsilon_C - WV^{-1}\epsilon_P \tag{3.15}$$

et

$$\Delta_P = V^{-1}(\epsilon_P - W^T\Delta_C) \tag{3.16}$$

où $(U - WV^{-1}W^T)$ est appelée la matrice de Schur de dimension $N_C \times N_C$ et notée S. Le système 3.14 possède des propriétés justifiant l'utilisation du complément de Schur :

- la construction de la matrice S dépend d'un produit matrice par matrice dont la complexité est de $\Theta(N_C^2)$,
- la complexité algorithmique pour résoudre Δ_C (équation 3.15) est de $\Theta(N_C^3)$ dans le cas où S est dense,
- résoudre Δ_P (équation 3.16) nécessite d'inverser V.

Sachant que l'inversion de V est triviale, la complexité du problème est de l'ordre de $\Theta(N_C^2 + N_C^3)$. Or le nombre de caméras est généralement faible comparativement aux points 3D. C'est pourquoi, l'utilisation du complément de Schur permet une diminution importante des temps de calcul pour des problèmes de reconstruction 3D.

3.4 Conclusion

Les différences entre l'algorithme de SLAM présenté dans ce chapitre et les méthodes de référence dans le domaine [84, 3] sont :

1. à l'initialisation : sélection du triplet d'images et calcul de la géométrie en se basant sur la méthode de Lui et Drummond [73],
2. à la localisation : utilisation d'un moindres carrés pondérés itératifs [78] pour gérer implicitement les *outliers*,
3. à la mise à jour de la carte : sélection différente des images clés et des paramètres à optimiser.

De nombreuses améliorations peuvent encore être apportées. Par exemple, Klein et Murray [85] utilisent des segments 3D, correspondant dans l'image aux contours, en complément des points 3D. Dans [2], Mur-Artal *et al.* utilisent le descripteur ORB qui est plus performant que la ZNCC. Cela permet de limiter la dérive du SLAM grâce à une meilleure propagation des amers visuels. De plus, l'utilisation d'un *kd-tree*[1] peut accélérer de façon significative les calculs d'appariements lors des étapes de re-localisation où l'image courante doit être appariée avec l'ensemble des descripteurs contenus dans la base. Il est également possible d'utiliser des méthodes de bouclage en-ligne afin de corriger la dérive du SLAM lorsque la trajectoire repasse dans une zone déjà cartographiée (la détection de boucle est là aussi réalisée avec un *kd-tree*).

1. Arbre de recherche multi-dimensionnelle utilisé pour partitionner un espace.

Chapitre 4

Résultats

Ce chapitre présente 3 expérimentations effectuées avec l'algorithme de SLAM détaillé dans cette partie. L'objectif est d'illustrer les performances de l'approche, en matières de temps de calcul et de dérive (liée à l'accumulation d'erreurs), sur des données réelles acquises en environnement intérieur et extérieur. Les séquences sont pré-enregistrées avec 3 caméras différentes, à des fréquences et des résolutions différentes. Les vidéos obtenues sont parcourues image par image par l'algorithme de SLAM. C'est pourquoi les temps de reconstruction sont inférieurs aux temps nécessaires à l'enregistrement des séquences. Pour chacune des expérimentations, la caméra utilisée est *global shutter*, le détecteur de points d'intérêt est celui de Harris [1] et les descripteurs sont de type ZNCC[1]. La plate-forme de test est un ordinateur de bureau équipé d'un processeur i7 cadencé à 3.4 MHz. Un seul processus système est utilisé donc les étapes de localisation et de mise à jour s'effectuent séquentiellement afin de simplifier l'interprétation des temps de calcul [2]. Les étapes de l'algorithme faisant appel à des méthodes d'optimisation non-linéaire (lors de l'initialisation, de la localisation, de la mise à jour de la carte et du bouclage) sont réalisées avec la bibliothèque LMA présentée dans la partie II du manuscrit. Les AF utilisés pour la localisation et la mise à jour de la carte sont limités à un maximum de 20 itérations et s'arrêtent lorsque la variation de l'erreur entre deux itérations est inférieure à $1e^{-4}\%$. En général, entre 2 et 5 itérations sont nécessaires pour la mise à jour de la carte, et entre 1 et 10 pour la localisation (en fonction du mouvement de la caméra).

4.1 Séquence Bureau

La figure 4.1 illustre une reconstruction effectuée en intérieur. L'intégralité de la séquence s'effectue dans une même pièce. Elle est constituée de 4300 images de résolution 640×480 acquises à la fréquence de 60 images par seconde avec une caméra portée à la main. Le calcul de pose est effectué avec le M-Estimateur robuste German-McClure présenté dans la section 3.2.2. Lors de la mise à jour de la carte, le nombre d'images clés optimisées est limité à un maximum de 5. Le détecteur de Harris est configuré pour extraire 300 points par image. 23 secondes sont nécessaires à la reconstruction de cette séquence de 173 images clés, 16K points 3D avec 59K points d'intérêt. Cela correspond à une vitesse de reconstruction d'approximativement 190 images par seconde. Les temps de traitement sont détaillés dans le tableau 4.1a. Pour une meilleure interprétation des résultats, l'étape de détection des points d'intérêt est dissociée de la localisation. On remarque que les étapes d'acquisition d'images et

1. Dont l'implémentation est disponible dans la libv : http://git.univ-bpclermont.fr/libv/libv
2. Si deux processus effectuent en parallèle un traitement d'une seconde, doit-on considérer que le temps de traitement est de 1 ou de 2 secondes ? la durée est effectivement d'une seconde mais la quantité d'instructions, et la consommation énergétique, équivalent à deux secondes de calcul. Dans ce travail, on veut montrer que peu de calculs sont requis (par opposition aux approches nécessitant un GPU pour atteindre le temps réel), mais une mesure en nombre de cycles processeurs peut être difficile à interpréter. C'est pourquoi, on choisit d'exprimer les résultats en temps de calcul et en utilisant un unique processus pour l'ensemble du traitement.

de détection avec le détecteur Harris sont les étapes qui nécessitent le plus de temps. Des images de la séquence, ainsi que le résultat de la reconstruction sont présentés dans la figure 4.1. Afin d'évaluer la dérive, on utilise un objet planaire statique référencé manuellement dans l'espace 3D. On observe alors que la projection de l'objet dans l'image tend à dériver au cours de la séquence. Cette dérive est liée à l'accumulation d'erreurs des méthodes de SLAM monoculaire et à la mauvaise propagation du facteur d'échelle.

4.2 Séquence Pavin

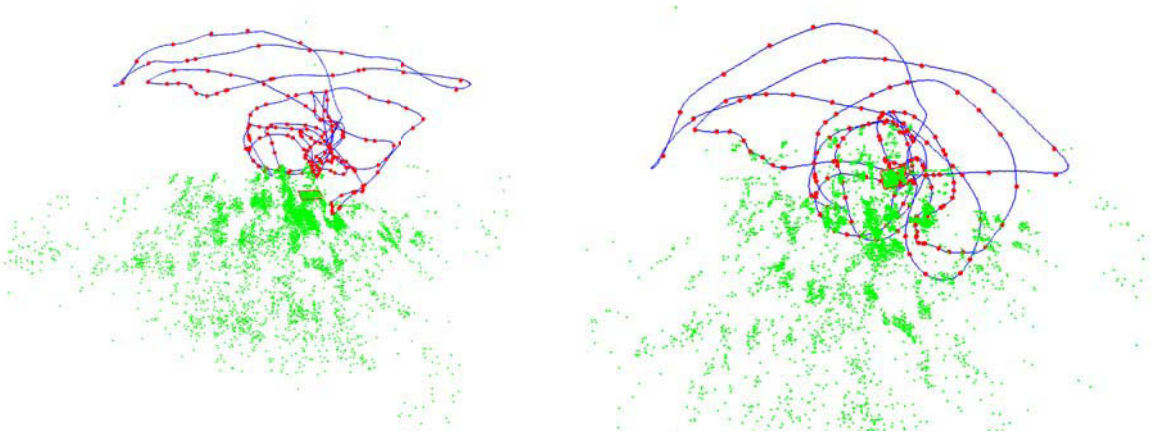
La figure 4.2 illustre une reconstruction effectuée sur la plate-forme PAVIN. La trajectoire mesure approximativement 250 mètres. Elle est constituée de 2392 images de résolution 1024×768 acquises à la fréquence de 7.5 images par seconde sur un VipaLab (véhicule électrique fabriqué par la société Apogée). L'algorithme est configuré de manière identique à l'expérimentation de la séquence "Bureau", mis à part le détecteur de Harris qui extrait ici 400 points par image (car la résolution d'image est plus importante). Le temps nécessaire à la reconstruction de cette séquence est de 20 secondes afin de reconstruire 257 images clés, 5796 points 3D et 99K points d'intérêt. Cela correspond à une vitesse de reconstruction d'approximativement 120 images par seconde. Les temps de traitement sont détaillés dans le tableau 4.2a. Pour une meilleure interprétation des résultats, l'étape de détection est dissociée de la localisation. On remarque comme précédemment que la détection des points d'intérêt avec le détecteur Harris est l'étape qui nécessite le plus de temps. Des images de la séquence, ainsi que le résultat de la reconstruction sont présentés dans la figure 4.2c. On observe que la trajectoire reconstruite (images de gauche) a dérivé. Cette dérive est inhérente à l'accumulation d'erreurs du SLAM. Pour corriger la dérive, on utilise la méthode de bouclage hors-ligne développée par Lébraly [86]. Le résultat du bouclage est illustré dans la figure 4.2 par la vue 3D à droite. Le temps nécessaire au bouclage de cette séquence est de 16 secondes, dont 3 secondes pour l'optimisation globale.

4.3 Séquence Cézeaux

La figure 4.3 illustre une reconstruction effectuée sur le campus universitaire des Cézeaux. La trajectoire mesure approximativement 3.4 km. Elle est constituée de 18279 images de résolution 752×480 acquises à la fréquence de 20 images par seconde sur un véhicule urbain. Le calcul de pose est effectué avec l'algorithme de RANSAC, présenté dans la section 3.2.2, qui est ici plus approprié aux conditions difficiles (caméra disposée derrière un pare-brise avec par moment beaucoup de végétations donc des points d'intérêt sur les arbres...). Lors de la mise à jour de la carte, le nombre d'images clés optimisées est limité à un maximum de 15. Le détecteur de Harris est configuré pour extraire 1300 points par image de manière répartie (en utilisant la méthode d'extraction des points d'intérêt par zone de Royer [31]). Le temps nécessaire à la reconstruction de cette séquence est de 3 minutes afin de constituer une carte de 2056 images clés, 47K points 3D et 310K points d'intérêt. Cela correspond à une vitesse de reconstruction d'approximativement 100 images par seconde. Les temps de traitement sont détaillés dans le tableau 4.3a. On utilise l'algorithme de fermeture de boucle décrit précédemment (*cf.* 4.2) pour corriger la dérive de la reconstruction. Le nombre de paramètres à optimiser étant ici très important, on utilise la méthode de complément de Schur Implicite de la bibliothèque LMA (présentée partie II) qui permet une réduction de la complexité algorithme de la résolution des équations normales lorsque le nombre de poses est important 6.3.1. Le résultat du bouclage est illustré dans la figure 4.3c par la vue 3D à droite. Le temps nécessaire au bouclage de cette séquence est de 6 minutes, dont 2 minutes pour l'optimisation globale.

	Acquisition	Détection	Localisation	Mise à jour	Total
Temps total (s)	11	10	1.5	0.5	23
Temps/image (ms)	2.5	2.3	0.3	2.8*	5.3

(a) Temps des différentes étapes de l'algorithme de SLAM sur la séquence Bureau composée de 4300 images, dont 173 sont des images clés. Sur cette séquence, le temps d'acquisition inclut la conversion de l'image du format bayer en niveau de gris. (*) : le temps moyen de la mise à jour est calculé sur le nombre d'images clés et non pas sur le nombre d'images total.



(b) Résultat de reconstruction sur la séquence Bureau (4300 images) : en 3D, le résultat de l'algorithme de SLAM (en bleu les calculs de pose sur l'ensemble des images, en rouge les images clés, et en vert les points 3D).

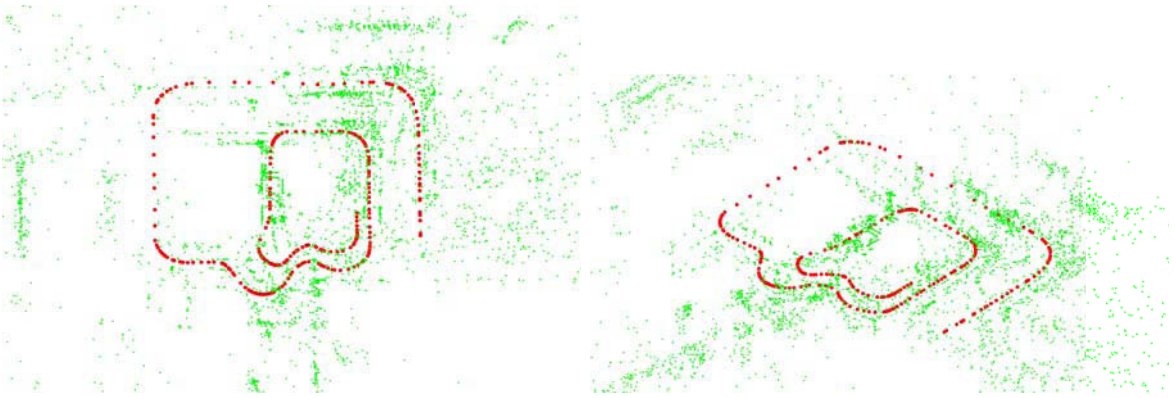
(c) Projection d'un objet planaire positionné manuellement dans le repère monde afin d'observer la dérive du SLAM au cours du temps.

FIGURE 4.1 – Expérimentation du SLAM en environnement intérieur.

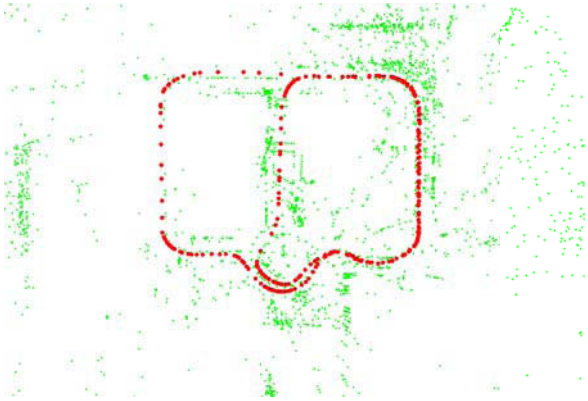
	Acquisition	Détection	Localisation	Reconstruction	Total
Temps total (s)	4	14	1	1	20
Temps/image (ms)	1.6	5.8	0.4	3.8*	8.3

(a) Temps des différentes étapes de l'algorithme de SLAM sur la séquence PAVIN composée de 2392 images, dont 257 sont des images clés. (*) : le temps moyen de la mise à jour est calculé sur le nombre d'images clés et non pas sur le nombre d'images total.

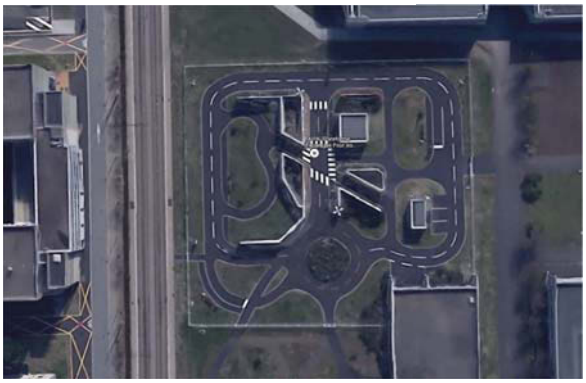
(b) Échantillon d'images issu de l'enregistrement de 2392 images.



(c) Reconstruction de l'algorithme de SLAM avec 257 images clés et 5796 points 3D.



(d) Reconstruction après bouclage hors-ligne.



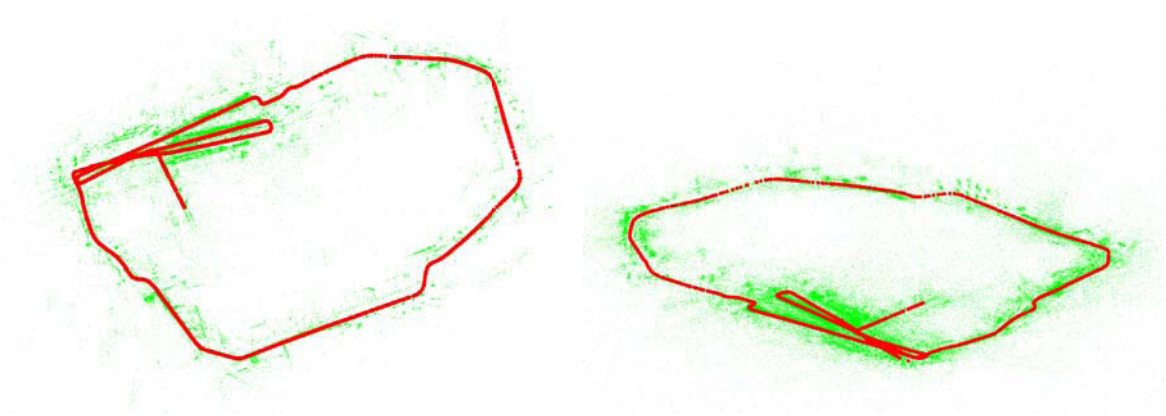
(e) Vue de PAVIN avec Google Map.

FIGURE 4.2 – Expérimentation du SLAM en environnement extérieur avec un véhicule électrique sur la plateforme PAVIN.

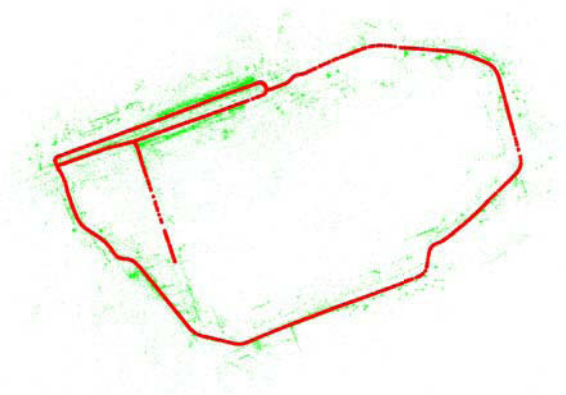
	Acquisition	Détection	Localisation	Reconstruction	Total
Temps total (s)	35	91	36	12	174
Temps/image (ms)	1.9	4.9	1.9	5.8*	9.5

(a) Temps des différentes étapes de l'algorithme de SLAM sur la séquence Cézeaux composée de 18279 images, dont 2056 sont des images clés. (*) : le temps moyen de la mise à jour est calculé sur le nombre d'images clés et non pas sur le nombre d'images total.

(b) Échantillon d'images issu de l'enregistrement de 18279 images.



(c) Reconstruction de l'algorithme de SLAM avec 2056 images clés et 47K points 3D.



(d) Reconstruction après bouclage hors-ligne.

(e) Vue des Cézeaux avec Google Map.

FIGURE 4.3 – Expérimentation du SLAM en environnement semi-urbain.

4.4 Conclusion

Les expérimentations ont démontré la possibilité d'utiliser l'algorithme de SLAM présenté dans des environnements intérieurs et extérieurs. Toutefois, les reconstructions subissent une dérive du facteur d'échelle, comme tout SLAM monoculaire. Il est possible de corriger la dérive en utilisant un algorithme de fermeture de boucle hors-ligne. Plusieurs approches temps-réel de fermeture de boucle dans un algorithme de SLAM par AF ont été proposées [1, 2]. Toutefois, cela n'est possible que si la trajectoire repasse dans une zone déjà cartographiée. Les étapes de localisation et reconstruction (incluant les phases d'appariements) représentent 10 à 30% du temps total de l'algorithme. L'étape qui nécessite le plus de temps est la détection des points d'intérêt avec le détecteur de Harris. Ainsi, un gain de temps de calcul important peut être fait en utilisant une implémentation plus efficace de cet algorithme, voire en utilisant le détecteur FAST développé par Rosten *et al.*[87]. Le détecteur FAST est plus rapide que le détecteur de Harris mais il est en général moins précis sur la qualité des détections. Le détecteur FAST demeure toutefois une alternative de plus en plus utilisée dans le domaine.

Conclusion

Nous avons proposé un algorithme de SLAM dont le principal avantage est un bon compromis entre robustesse et vitesse d'exécution. Comme tout autre algorithme de SLAM monoculaire, on observe des dérives de la localisation relativement importantes dues à l'accumulation d'erreurs. Pour limiter cette dérive, il semble inévitable d'introduire des informations supplémentaires au SLAM monoculaire. On parle alors de SLAM contraint. C'est pourquoi, l'algorithme de SLAM proposé ici constitue une base de travail intéressante pour les expérimentations futures où des contraintes sont ajoutées au processus d'optimisation afin de corriger cette dérive.

Deuxième partie

Une bibliothèque pour l'optimisation : LMA

Il existe de nombreuses implémentations des méthodes de moindres carrés non linéaires sous forme de logiciels ou de bibliothèques. Celles-ci sont écrites dans divers langages informatiques et offrent des performances souvent inversement proportionnelles à la facilité d'utilisation.

Pour résoudre des problèmes d'optimisation variés, les outils de l'état de l'art utilisent des implémentations abstraites : algorithmes écrits dans un langage de programmation haut niveau utilisant des mécanismes d'abstraction (les notions d'implémentations abstraites et spécialisées sont détaillées dans la section 1.12.3). C'est pourquoi, il est possible de proposer des implémentations, dites spécialisées, plus efficaces pour résoudre un problème particulier simplement en implémentant l'algorithme de résolution spécifiquement pour le problème donné. On distingue deux approches pour spécialiser un code informatique : en utilisant des instructions bas niveau spécifiques à l'architecture matérielle, ou en spécialisant l'implémentation de l'algorithme pour la résolution d'un unique problème. Toutefois, les implémentations spécialisées perdent en généricité et les instructions bas niveau rendent le code difficile à maintenir. Pour résoudre une plus grande variété de problèmes, on peut utiliser des codes abstraits dont l'implémentation est proche de l'écriture algorithmique de la solution, mais cela se fait au détriment des performances. La difficulté principale dans la conception d'outils de haut niveau pour les problèmes d'optimisation est donc de proposer une méthode généraliste qui allie facilité d'utilisation et performances. Ceci explique pourquoi peu d'outils ont été développés.

Nous proposons ici une nouvelle implémentation nommée LMA de l'algorithme de Levenberg-Marquardt sous la forme d'une bibliothèque *open-source* développée en C++ afin de tirer parti des évolutions récentes en termes de langage de programmation et de compilation. L'objectif de cette implémentation est d'être à la fois plus simple d'utilisation et plus performante que les alternatives de l'état de l'art.

Après une brève présentation des bibliothèques d'optimisation existantes dont la majorité est généraliste, nous aborderons dans cette partie la problématique liée à une implémentation efficace d'un algorithme d'optimisation. Un intérêt particulier sera porté à la nature éparse et l'implémentation de problèmes d'ajustement de faisceaux. En réalité, il n'est pas possible de déterminer une structure de données qui soit efficace pour tous les problèmes. Ceci nous amène à la solution proposée : concevoir un méta-programme (LMA) capable d'analyser les caractéristiques d'un problème d'optimisation pour générer le code d'un solveur spécialisé pour la résolution de ce problème en particulier. LMA se compose de deux types d'algorithme : 1- des algorithmes qui analysent du code source et génèrent une structure de données et un solveur pendant le processus de compilation, 2- des algorithmes qui utilisent le solveur pour résoudre le problème donné à l'exécution du programme. Ensuite, on présentera les différentes fonctionnalités de LMA que l'on retrouve couramment dans les bibliothèques d'optimisation (résolution d'équations linéaires, complément de Schur, méthodes de dérivées, méthodes robustes...) pour finir par une série d'expérimentations.

Chapitre 5

Algorithme de Levenberg-Marquardt et implémentations

On parle d'optimisation pour nommer les méthodes de moindres carrés non linéaires utilisées pour estimer un ensemble de paramètres réels minimisant une fonction d'erreur (la section 1.5 présente avec plus de détails les méthodes d'optimisation). Parmi les différents algorithmes existants, on se focalisera sur l'algorithme de Levenberg-Marquardt qui est une méthode robuste en cas de mauvaise initialisation et lorsque les erreurs sur les données sont importantes. La section qui suit présente différents solveurs implémentant l'algorithme de Levenberg-Marquardt pour la résolution de problèmes d'optimisation, dont certains ont des implémentations spécialisées pour la résolution de problèmes éparses comme par exemple l'ajustement de faisceaux (AF).

5.1 Outils et bibliothèques

De nombreuses implémentations de l'algorithme de Levenberg-Marquardt sont disponibles à travers divers outils et langages de programmation : Mathematica, Matlab, Scilab, MinPack, Eigen, NT2, Toon, Armadillo, Blaze ... Ces implémentations se veulent à la fois généralistes et simples d'utilisation. Le plus souvent, elles utilisent des matrices denses, donc limitées à un nombre restreint de paramètres. Les outils basés sur des matrices éparses sont plus rares car la structure éparse découle directement de la nature du problème à résoudre. Ainsi, il n'est pas possible de définir une structure de matrice éparse généraliste qui soit adaptée à n'importe quel type de problème d'optimisation. C'est pourquoi, il existe des bibliothèques spécialisées pour les problèmes d'AF.

On présente par la suite quelques bibliothèques *open-source* utilisables pour résoudre des problèmes d'AF. Elles utilisent le langage de programmation C++ pour des raisons de performance. De plus, elles utilisent plus ou moins d'*a priori* sur la structure éparse spécifique à l'AF comme par exemple l'utilisation du complément de Schur.

Bibliothèque sba [88]

Développée par Lourakis *et al.* en 2004, sba est l'une des premières bibliothèques populaires pour résoudre des problèmes d'AF. Cette bibliothèque est générique sur le système de coordonnées, la paramétrisation et la fonction de coût. Son implémentation des équations normales est partiellement éparse : une méthode de Cholesky dense du paquet LAPACK [1] est utilisée pour résoudre les équations normales ce qui limite son usage aux problèmes de petites et moyennes tailles. De plus, adapter cette bibliothèque pour de nouveaux problèmes d'AF peut être difficile comparativement aux bibliothèques plus récentes.

1. bibliothèque d'algèbre linéaire développée en Fortran.

Framework g2o [89]

g2o est un *framework* développé par Kummerle *et al.* pour traiter des problèmes d'optimisation non linéaire modélisés par des graphes. Cette bibliothèque utilise une structure éparse basée sur des blocs de matrices de taille fixe (la dimension de chacun des blocs est définie à la compilation) pour une résolution efficace dans des configurations à deux familles de paramètres (pose et point 3D). Afin de résoudre de multiples problèmes d'AF, ce framework possède un mécanisme d'extension basé sur l'héritage pour gérer différents types de familles de paramètres et de fonctions d'erreurs. D'après le comparatif publié dans [89], les performances de la bibliothèque g2o sont meilleures que la bibliothèque sba pour des problèmes de petites et moyennes dimensions.

Framework GTSAM [90]

La bibliothèque GTSAM, pour (*Georgia Tech Smoothing and Mapping*), est développée par Dellaert *et al.* [90] pour résoudre des problèmes d'optimisation éparse en utilisant une représentation en *factor graphs*. Cela consiste à connecter, au sein d'un graphe de paramètres, des densités de probabilité obtenues grâce à des observations ou des connaissances *a priori*. Cette bibliothèque permet de résoudre des problèmes d'estimation de paramètres à travers une approche probabiliste et elle est utilisée pour des applications tout à fait semblables aux autres outils présentés ici (comme des problèmes de reconstruction 3D). Des comparatifs positionnent la bibliothèque GTSAM au même niveau de performance que la bibliothèque g2o [91, 92].

Programme pba [93]

Jusqu'à maintenant, pba est le programme le plus optimisé et spécialisé pour la résolution de gros problèmes d'AF. Elle est développée par Wu *et al.* avec l'objectif d'exploiter les possibilités offertes par les architectures parallèles comme le SIMD [2], le *multi-thread* [3] et le GPU [4]. Pour résoudre des problèmes impliquant des milliers de caméras et des millions de points 3D, pba utilise la méthode du complément de Schur implicite qui est présentée dans la sous-section 6.3.1. Toutefois, le code est spécialisé pour un certain type d'architecture et pour un type d'AF en particulier. L'utilisation de cette bibliothèque n'est donc pas envisageable pour résoudre un problème d'optimisation différent.

Bibliothèque Ceres [94]

Ceres est certainement la bibliothèque *open-source* la plus avancée pour résoudre des problèmes d'optimisation. Elle est développée par Agarwal *et al.*, supportée par Google, et utilise des bibliothèques performantes d'algorithmes éparses d'algèbre linéaire. Plusieurs stratégies sont implémentées comme la minimisation par région de confiance. Elle permet l'utilisation du complément de Schur implicite. Elle contient également une collection de solveurs linéaires denses et éparses, basés sur Eigen [95] et sur les bibliothèques SuiteSparse [96]. Ceres peut gérer des fonctions de coût à plusieurs termes avec différentes familles de paramètres, elle est donc adaptée à la résolution d'un grand nombre de problèmes. De plus, grâce à une interface simple, de bonnes performances et une licence libre, cette bibliothèque est de plus en plus utilisée dans de multiples champs d'applications.

5.2 Problématique

Aucune des solutions présentes dans l'état de l'art ne permet de résoudre efficacement tous les problèmes d'optimisation. Certaines sont spécialisées pour les moyens et gros problèmes (Ceres), d'autres

2. *Single Instruction Multiple Data* : instructions vectorielles opérant simultanément sur plusieurs flux de données.
3. Utilisation de plusieurs fils d'exécution au sein d'un même processus système.
4. *Graphics Processing Unit* : unité de calcul présent sur les cartes graphiques.

pour les problèmes de moyenne taille (g2o) et d'autres pour les problèmes denses [5] (Eigen, NT2, TooN, Blaze...). Les plus efficaces ne permettent de résoudre qu'un unique problème (pba). Parmi l'ensemble des bibliothèques présentées, seulement trois offrent un niveau de généricité suffisant pour les problèmes d'optimisation qu'on rencontrera dans la suite de ce travail de thèse. Il s'agit de gtsam, g2o et Ceres. Ces bibliothèques utilisent le polymorphisme du langage C++ pour permettre la gestion de plusieurs fonctions d'erreur au sein d'une même optimisation. Ces fonctions étant intensivement utilisées lors de la résolution pour évaluer le coût de la fonction et/ou calculer les dérivées, l'utilisation du polymorphisme nuit aux performances (*cf.* 1.12.2). De plus, les contraintes liées à l'utilisation de ces bibliothèques peuvent s'avérer bloquantes. Par exemple, le nombre de familles [6] de paramètres à optimiser au sein d'un même problème est limité :

— à 2 pour g2o, lorsque qu'on utilise la méthode de résolution éparse par blocs de taille fixe (qui est la méthode la plus performante),
— à 6 pour gtsam,
— à 10 pour ceres.

L'utilisation de g2o implique d'inclure les paramètres à optimiser et les observations dans la hiérarchie de classe du *framework* (via l'héritage), ce qui constitue une contrainte forte sur l'implémentation des algorithmes (la structure de données du problème doit être spécifiquement conçue pour l'utilisation de g2o). Ceres restreint le type des paramètres à optimiser à des tableaux de valeurs flottantes et ne permet pas de définir simplement des méthodes de mise à jour de paramètres (comme le fait g2o).

5.3 Difficultés liées à l'implémentation d'un ajustement de faisceaux

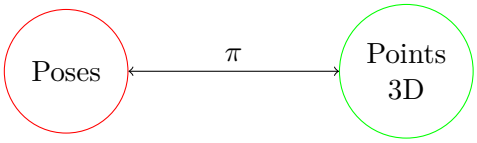
Il est difficile d'implémenter un algorithme d'optimisation qui soit tout le temps efficace car chaque problème nécessite une structure de données et une implémentation différentes. Dans cette section, on montre comment déterminer la structure interne des équations normales, et donc la structure de données correspondante, en fonction de la fonction de coût de deux problèmes d'AF relativement similaires. L'objectif est d'illustrer que la résolution efficace de chacun des problèmes nécessite une structure de données différente, et que celle-ci est potentiellement complexe.

5.3.1 Problème d'ajustement de faisceaux à deux familles de paramètres

Considérons un problème classique d'ajustement de faisceaux (AF) où les poses sont paramétrées avec 6 degrés de liberté (*ddl*) $\alpha, \beta, \gamma, Tx, Ty, Tz$ (3 pour l'orientation et 3 pour la position) et les points 3D avec 3 degrés *ddl.* Px, Py, Pz. La fonction de coût $\mathcal{E}$ à minimiser correspond à la somme des K erreurs de reprojection :

$$\mathcal{E} = \sum_{k=0}^{K} \parallel \pi(C_{p_k}, P_{p_k}) - p_k \parallel^2 \tag{5.1}$$

D'après la fonction $\mathcal{E}$, les paramètres de poses et de points 3D sont liés : si l'on modifie les paramètres d'une pose, l'erreur de reprojection d'un point 3D observé dans cette pose sera également modifiée. Ainsi, on définit une représentation du problème sous forme de graphe telle que les paramètres de poses et de points 3D soient liés par la fonction de coût π :



5. où tous les paramètres sont liés entre eux ce qui nécessite l'utilisation de matrices denses.
6. Une famille de paramètres correspondant à l'ensemble des paramètres d'un même type.

Cette représentation sera re-utilisée par la suite pour modéliser les relations entre paramètres pour des problèmes plus complexes.

Calcul de la jacobienne

Pour obtenir la jacobienne J de la fonction $\mathcal{E}$, il faut calculer la dérivée de chacune des K erreurs de reprojection en fonction de la pose et du point 3D associé. La jacobienne J est alors composée d'un ensemble de « petites » jacobiennes, aussi appelées « blocs », correspondant à la dérivée des 9 paramètres $\alpha, \beta, \gamma, Tx, Ty, Tz, Px, Py, Pz$ en fonction des deux dimensions d'un point d'intérêt (u, v). Les blocs sont donc composés des 18 valeurs suivantes :

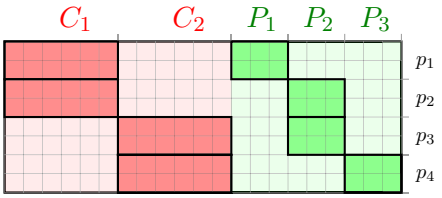
$\frac{\partial u}{\partial \alpha}$	$\frac{\partial u}{\partial \beta}$	$\frac{\partial u}{\partial \gamma}$	$\frac{\partial u}{\partial T_x}$	$\frac{\partial u}{\partial T_y}$	$\frac{\partial u}{\partial T_z}$	$\frac{\partial u}{\partial P_x}$	$\frac{\partial u}{\partial P_y}$	$\frac{\partial u}{\partial P_z}$
$\frac{\partial v}{\partial \alpha}$	$\frac{\partial v}{\partial \beta}$	$\frac{\partial v}{\partial \gamma}$	$\frac{\partial v}{\partial T_x}$	$\frac{\partial v}{\partial T_y}$	$\frac{\partial v}{\partial T_z}$	$\frac{\partial v}{\partial P_x}$	$\frac{\partial v}{\partial P_y}$	$\frac{\partial v}{\partial P_z}$

En pratique, on regroupe de manière ordonnée les paramètres de poses et les paramètres de points 3D. C'est pourquoi, la jacobienne J est constituée de 2 parties :

$$J = \begin{bmatrix} J_c & J_p \end{bmatrix} \tag{5.2}$$

telles que
- J_c contienne l'ensemble des blocs de poses (en rouge),
- J_p contienne l'ensemble des blocs de points 3D (en vert).

A titre d'exemple, la jacobienne pour une configuration à 2 poses C_1 et C_2 et 3 points 3D P_1, P_2 et P_3 (P_1 est vu dans C_1, P_2 est vu dans C_1 et C_2, et P_3 est vu dans C_3) est de la forme suivante :



où les dérivées non nulles sont représentées en couleur foncée, les dérivées nulles en couleur claire. La jacobienne est fortement éparse pour deux raisons :
- les points 3D ne sont pas observés dans toutes les poses,
- une observation p_k ne va avoir un impact que sur 9 paramètres (une pose et un point 3D).

Ainsi, chacune des lignes de la jacobienne ne contient que 9 valeurs non nulles, quel que soit le nombre de poses et points 3D du problème.

Calcul de la hessienne

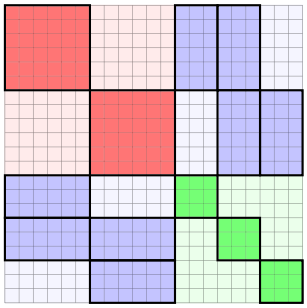
Pour l'algorithme de Levenberg-Marquardt, le système d'équations linéaires à résoudre est le suivant :

$$(J^T J + \lambda I)\Delta = J^T \epsilon \tag{5.3}$$

La hessienne H (approximée par $J^T J$) est constituée de 4 sous-matrices éparses : $H_{cc} = J_c^T J_c + \Lambda I$, $H_{cp} = J_c^T J_p$, $H_{pc} = J_p^T J_c$ et $H_{pp} = J_p^T J_p + \lambda I$:

$$H = J^T J = \left[\begin{array}{c|c} H_{cc} & H_{cp} \\ \hline H_{pc} & H_{pp} \end{array}\right] \tag{5.4}$$

En reprenant la jacobienne pris en exemple précédemment, la hessienne est alors de la forme suivante :



avec en rouge la hessienne des poses, en vert la hessienne des points 3D, et en bleu les dérivées croisées entre poses et points 3D.

Complément de Schur

Avec la hessienne précédemment calculée, le système linéaire suivant

$$\begin{bmatrix} H_{cc} & H_{cp} \\ H_{pc} & H_{pp} \end{bmatrix} \begin{bmatrix} \Delta_c \\ \Delta_p \end{bmatrix} = \begin{bmatrix} J_c^T \epsilon_c \\ J_p^T \epsilon_p \end{bmatrix}$$

est résolu en utilisant le complément de Schur (présenté dans la section 3.11) :

$$\begin{cases} (H_{cc} - H_{cp} H_{pp}^{-1} H_{pc}) \Delta_c = J_c^T e_c - H_{cp} H_{pp}^{-1} J_p^T \epsilon_p \\ \Delta_p = H_{pp}^{-1} (J_p^T \epsilon_p - H_{pc} \Delta_c) \end{cases}$$

La structure du système d'équations linéaires d'un problème classique d'AF possède différentes caractéristiques (en tenir compte permet d'améliorer l'implémentation du problème) :

- chaque ligne de la jacobienne ne contient que 9 valeurs non nulles, quel que soit le nombre de paramètres du problème,
- les paramètres des poses sont indépendants entre eux,
- idem pour les points 3D,
- c'est pourquoi, H_{cc} et H_{pp} sont éparses, symétriques, diagonales par blocs et donc facilement inversibles,
- H_{cp} est éparse par blocs et $H_{cp} = H_{pc}^T$,
- Δ et $J^T \epsilon$ sont des vecteurs colonnes.

Ainsi, la résolution efficace du système linéaire est conditionnée par la qualité de l'implémentation du complément de Schur, donc de la structure de données utilisée pour représenter les matrices éparses par blocs H_{cc}, H_{pp} et H_{cp}. A l'intérieur de chacune de ces 3 sous-matrices, tous les blocs sont de même taille. On connaît les dimensions des blocs car celles-ci sont liées aux *ddl.* du problème. Ainsi, la structure H_{cc} est composée de blocs de dimension 6×6, H_{pp} de blocs de dimension 3×3, et H_{cp} par des blocs de dimension 6×3. Polok *et al.* [91] ont montré que l'implémentation efficace d'un tel problème nécessite l'utilisation d'une structure éparse par blocs, où la dimension de chacun des blocs est connue à la compilation. On présente une telle structure, nommée *SBCRS* pour *Static Blocs Compressed Row Storage*, dans l'annexe 11.5.3. Avec la représentation *SBCRS*, la structure de données correspondant à H est composée des 3 conteneurs suivants :

$$SBCRS(H) = \begin{Bmatrix} H_{cc} = BCRS_{6\times6} \\ H_{cp} = BCRS_{6\times3} \\ H_{pp} = BCRS_{3\times3} \end{Bmatrix} \tag{5.5}$$

La résolution de ce problème en C++ nécessite d'implémenter les opérateurs matriciels nécessaires au complément de Schur et à la résolution de systèmes linéaires pour la structure $SBCRS(H)$:

— addition,
— soustraction,
— produit matriciel,
— transposée,
— inversion de matrices symétriques diagonales par blocs.

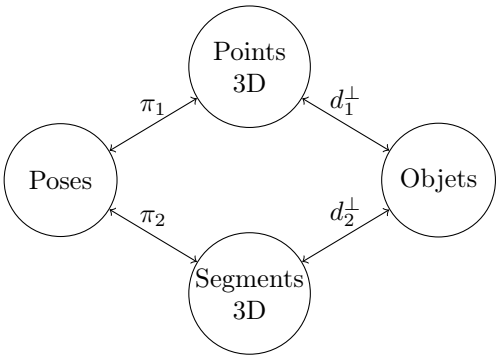
L'approche présentée ici constitue, pour l'ajustement de faisceaux, l'implémentation de référence utilisée dans la majorité des travaux [59, 67, 88, 89].

5.3.2 Problème à 4 familles de paramètres

On ajoute maintenant au problème d'optimisation précédent deux familles de paramètres : des segments 3D (amers visuels correspondant aux contours dans les images) et des objets 3D présents dans l'environnement. Ce nouveau problème implique des erreurs de reprojection π_1 et π_2 entre les poses et les amers visuels de types points et segments 3D, ainsi que des erreurs de distance orthogonale $d_1^\perp$ et $d_2^\perp$ entre les objets 3D et amers 3D (les points et les segments 3D). Ce problème correspond typiquement à un SLAM avec deux types d'amers visuels ainsi que des objets 3D dont les paramètres de pose et de dimension sont conjointement estimés. La fonction de coût est la suivante (avec S3D et Obj correspondant respectivement à un segment et un objet 3D) :

$$\mathcal{E} = \|\pi_1(\text{Poses}, \text{P3D})\|^2 + \|\pi_2(\text{Poses}, \text{S3D})\|^2 + \|d_1^\perp(\text{Objets}, \text{P3D})\|^2 + \|d_2^\perp(\text{Objets}, \text{S3D})\|^2 \tag{5.6}$$

et peut être représentée par le graphe :



La jacobienne de la fonction de coût $\mathcal{E}$ est de la forme suivante :

$$J = \begin{bmatrix} J_c^{\pi_1} & 0 & J_p^{\pi_1} & 0 \\ J_c^{\pi_2} & 0 & 0 & J_s^{\pi_2} \\ 0 & J_o^{d_1^\perp} & J_p^{d_1^\perp} & 0 \\ 0 & J_o^{d_2^\perp} & 0 & J_s^{d_2^\perp} \end{bmatrix} \tag{5.7}$$

dont les parties non nulles correspondent aux paramètres associés aux fonctions d'erreurs :

	Poses	Objets	P3D	S3D
π_1	×		×	
π_2	×			×
$d_1^\perp$		×	×	
$d_2^\perp$		×		×

(5.8)

La structure éparse de la hessienne H, approximée par J^TJ, est alors de la forme :

$$H = J^TJ = \left[\begin{array}{cc|cc} H_{cc} & 0 & H_{cp} & H_{cs} \\ 0 & H_{oo} & H_{op} & H_{os} \\ \hline & & H_{pp} & 0 \\ & & 0 & H_{ss} \end{array}\right] \tag{5.9}$$

On remarque que les parties de la hessienne correspondant à H_{co} et H_{ps} sont nulles car d'après le graphe du problème, les paramètres de poses ne sont pas liés aux objets. Il en va de même pour les points et les segments 3D. Le complément de Schur (*cf.* 3.11) s'effectue avec $V = \begin{bmatrix} H_{pp} & 0 \\ 0 & H_{ss} \end{bmatrix}$. La condition telle que V soit diagonale par blocs est satisfaite car à la fois H_{pp} et H_{ss} sont diagonales par blocs.

Comme précédemment, les nombres de *ddl.* d'une pose et d'un point 3D sont respectivement de 6 et de 3. Pour cet exemple, on considère que les segments 3D et les objets possèdent respectivement 4 (paramétrisation minimale utilisée par Klein *et al.* [85]) et 9 *ddl.* (3 pour l'orientation, 3 pour la position et 3 pour les dimensions). La hessienne est composée de 8 structures différentes, ainsi elle peut être implémentée avec la représentation éparse $SBCRS$ sous la forme de 8 conteneurs (la hessienne étant symétrique, on peut n'implémenter que la moitié triangulaire supérieure) :

$$SBCRS(H) = \left\{\begin{array}{l} H_{cc} = BCRS_{6\times6} \\ H_{cp} = BCRS_{6\times3} \\ H_{cs} = BCRS_{6\times4} \\ H_{oo} = BCRS_{9\times9} \\ H_{op} = BCRS_{9\times3} \\ H_{cs} = BCRS_{6\times4} \\ H_{pp} = BCRS_{3\times3} \\ H_{ss} = BCRS_{4\times4} \end{array}\right\} \tag{5.10}$$

Il s'agit maintenant d'implémenter le complément de Schur pour la structure de données $SBCSR(H)$:

$$\left\{\begin{array}{l} \left(\begin{bmatrix} H_{o,o} & 0 \\ 0 & H_{c,c} \end{bmatrix} - \begin{bmatrix} H_{o,p} & H_{o,s} \\ H_{c,p} & H_{c,s} \end{bmatrix} \begin{bmatrix} H_{p,p} & 0 \\ 0 & H_{s,s} \end{bmatrix}^{-1} \begin{bmatrix} H_{o,p} & H_{c,p} \\ H_{o,s} & H_{c,s} \end{bmatrix}\right) \begin{bmatrix} \Delta_o \\ \Delta_c \end{bmatrix} = \\ \qquad\qquad \begin{bmatrix} J_o^T\epsilon_o \\ J_k^T\epsilon_c \end{bmatrix} - \begin{bmatrix} H_{o,p} & H_{o,s} \\ H_{c,p} & H_{c,s} \end{bmatrix} \begin{bmatrix} H_{p,p} & 0 \\ 0 & H_{s,s} \end{bmatrix}^{-1} \begin{bmatrix} J_p^T\epsilon_p \\ J_p^T\epsilon_s \end{bmatrix} \\ \begin{bmatrix} \Delta_p \\ \Delta_e \end{bmatrix} = \begin{bmatrix} H_{p,p} & 0 \\ 0 & H_{e,e} \end{bmatrix}^{-1} \left(\begin{bmatrix} J_p^T\epsilon_p \\ J_e^T\epsilon_e \end{bmatrix} - \begin{bmatrix} H_{o,p} & H_{k,p} \\ H_{o,e} & H_{k,e} \end{bmatrix} \begin{bmatrix} \Delta_o \\ \Delta_k \end{bmatrix}\right) \end{array}\right.$$

Sans même développer le calcul, on remarque que l'expression du système à résoudre est bien plus complexe que pour le problème précédent impliquant 2 familles de paramètres. En fait, la complexité d'implémentation augmente quadratiquement avec le nombre de familles de paramètres. Pour résoudre ce système d'équations en C++, il faut de nouveau implémenter les opérateurs matriciels ($+$,$-$,$\times$,...) pour la structure $SBCRS(H)$. Il y a donc une double problématique à l'implémentation d'algorithmes d'optimisation :

— la structure de données change d'un problème à un autre,
— la structure de données et l'implémentation des calculs deviennent rapidement compliquées lorsque le nombre de familles de paramètres augmente.

Dans la section suivante, on présente une solution à cette problématique.

5.4 Solution proposée

Dès lors que le problème à optimiser devient complexe, la difficulté d'implémentation et le temps de développement deviennent trop importants. Pourtant, la complexité algorithmique de la résolution reste liée au nombre de paramètres et non au nombre de familles de paramètres. C'est pourquoi, le point bloquant est davantage lié à la difficulté d'implémentation qu'au temps de calcul. Les solutions de l'état de l'art (*cf.* 5.1) peuvent résoudre ce type de problèmes mais cela se fait au détriment des performances, de la facilité d'utilisation et de la généricité. De plus, GTSAM, g2o et Ceres (qui sont actuellement les meilleurs outils pour résoudre ce type de problèmes) utilisent le mécanisme d'héritage du langage orienté objet C++ pour assurer un niveau suffisant de généricité. Cela nuit aux performances car l'utilisation du polymorphisme crée un niveau d'indirection supplémentaire dans les appels de fonctions et limite les possibilités d'optimisation du code par le compilateur.

La contribution proposée pour résoudre ce problème est de concevoir un méta-programme (un programme qui en écrit un autre) dont le rôle est de générer automatiquement un solveur basé sur l'algorithme de Levenberg-Marquardt quels que soit le nombre de familles de paramètres et de nombre de contraintes du problème. De plus, la solution proposée a pour ambition d'être plus simple d'utilisation, plus générique et plus performante que les bibliothèques de l'état de l'art. Ce méta-programme prend la forme d'une bibliothèque *open-source* implémentant l'algorithme de Levenberg-Marquard avec le langage C++ et nommée LMA pour (*Levenberg-Marquardt Algorithm*). Contrairement aux bibliothèques classiques, LMA ne résout pas des problèmes d'optimisation mais génère un code qui résout des problèmes d'optimisation. Ce processus de génération de code est effectué automatiquement, à travers le processus de compilation, en utilisant les concepts de méta-programmation et programmation générative présentés en annexe 1.12.4.

Chapitre 6

Implémentation de LMA

Ce chapitre décrit l'implémentation de l'algorithme de Levenberg-Marquard au sein de la bibliothèque LMA : *Levenberg-Marquardt Algorithm*. L'objectif de cette bibliothèque est de fournir une interface simple avec un processus non intrusif d'adaptation au problème tout en conservant de bonnes performances d'exécution pour des problèmes de petite, moyenne et grande dimensions. L'objectif est d'automatiser le processus d'écriture de l'algorithme de Levenberg-Marquardt afin de générer un solveur qui soit aussi performant qu'un code écrit à la main par une personne maîtrisant toutes les caractéristiques du problème. Après une présentation générale de la méthode, les algorithmes d'analyse du problème et de génération de la structure de données sont détaillés. Puis les différentes méthodes de résolution des systèmes d'équations linéaires sont présentées ainsi que certaines fonctionnalités courantes (calculs de dérivées) ou originales (généricité des calculs sur le type matriciel). Ce chapitre finit sur une présentation de l'API simple et minimaliste qui constitue un point fort de la bibliothèque.

6.1 Vue d'ensemble

LMA résout des problèmes d'optimisation exprimés sous la forme d'une somme de N fonctions de coût :

$$\mathcal{E} = \sum_{i=1}^{N} \mathcal{F}_i \tag{6.1}$$

Chacune des fonctions de coût $\mathcal{F}_i$ effectue un calcul de nature différente, composé d'un nombre N_i d'instances $F_{i,j}$ avec $1 \leq j \leq N_i$:

$$\mathcal{F}_i = \sum_{j=1}^{N_i} \|F_{i,j}\|^2 \tag{6.2}$$

où chaque instance correspond au calcul d'un terme de l'erreur (ou résidu). L'ensemble des instances $F_{i,j}$ d'une fonction $\mathcal{F}_i$ effectue le même calcul d'erreur mais avec des paramètres différents. La fonction $\mathcal{E}$ correspond à la somme des erreurs du problème, donc à la fonction de coût à minimiser :

$$\mathcal{E} = \sum_{j=1}^{N_1} \|F_{1,j}\|^2 + ... + \sum_{j=1}^{N_n} \|F_{n,j}\|^2 \tag{6.3}$$

Cette section décrit le fonctionnement de la bibliothèque LMA, résumé par la figure 6.1, qui se divise en deux temps :

1. algorithme CT (*compile-time*) : l'analyse du problème et la génération du solveur à la compilation,
2. algorithme RT (*run-time*) : la résolution du problème avec l'algorithme de Levenberg-Marquardt à l'exécution du programme.

A l'exécution du programme, le solveur généré par le processus de compilation est alimenté avec les foncteurs d'erreurs et les paramètres associés. Puis l'algorithme de Levenberg-Marquardt est utilisé pour résoudre le problème. Cet algorithme est implémenté afin d'utiliser la structure de données spécialisée qui a été générée à la compilation. L'algorithme étant présenté dans les notions de base (*cf.* algorithme 1), il ne sera pas rappelé dans ce chapitre.

6.2 Algorithme CT

L'objectif de LMA est de fournir une interface simple avec un processus non intrusif d'adaptation au problème tout en conservant de bonnes performances d'exécution. C'est pourquoi, LMA ne possède pas de liste prédéfinie de fonctions de coût ou de paramètres supportés, car cela limiterait son utilisation à un nombre restreint de problème. Les caractéristiques du problème -constituées des différents termes de la fonction de coût, des paramètres et de paramétrisation spécifiques (*ddl.* et mise à jour des paramètres)- sont définies par l'utilisateur et ne sont donc pas connues à l'avance. L'utilisation de ces informations étant nécessaire pour générer une implémentation efficace, il est nécessaire d'injecter ces données à la bibliothèque via une étape de configuration. Toutefois, une telle procédure peut ajouter de la lourdeur à l'utilisation et peut en plus être source d'erreur si le problème est complexe. C'est pourquoi, l'originalité de LMA réside dans sa capacité à s'auto-configurer pendant le processus de compilation en analysant les différentes caractéristiques du problème. Pour cela, on a recours à des techniques de programmation avancée (*cf.* méta-programmation 1.12.4) pour analyser le code source du problème défini par l'utilisateur afin d'exploiter un maximum d'informations disponibles à la compilation. L'objectif est de générer un solveur spécialisé pour le problème donné.

L'algorithme CT est illustré par la figure 6.1 qui résume les différentes étapes décrites dans cette section. Les entrées de cet algorithme sont de trois types :

1. la liste des fonctions d'erreur,
2. les degrés de liberté pour chaque type de paramètre,
3. les méthodes de mise à jour des paramètres.

Les caractéristiques du problème sont obtenues à partir d'une liste $L = F_1, ..., F_n$ de foncteurs C++ définissant la fonction d'erreur $\mathcal{E}$ (*cf.* 6.2.1). Les paramètres sont déduits à partir des arguments des foncteurs (*cf.* 6.2.2) et les relations entre paramètres sont détectées automatiquement afin de générer un graphe de relations (*cf.* 6.2.3) dans le but de définir la structure de données du problème (*cf.* 6.2.4). Puis les équations normales ($H\Delta = J^T\epsilon$) sont générées en utilisant pour chaque type de paramètres les *ddl.* définis (*cf.* 6.2.5). Puis le solveur implémentant l'algorithme de Levenberg-Marquardt est généré avec les méthodes de résolution de système linéaire requises (*cf.* 6.3).

Il faut noter que l'ensemble des algorithmes présentés dans cette section s'exécute à la compilation du programme. Ainsi la complexité algorithmique n'a ici aucune influence sur les performances d'exécution. De plus, l'écriture de ces algorithmes avec un méta-langage (ici la méta-programmation *template* C++) peut s'avérer complexe. C'est pourquoi, on privilégie la simplicité d'écriture des algorithmes à la complexité algorithmique.

6.2.1 Les foncteurs

La liste des fonctions d'erreurs $L = F_1, ..., F_n$ correspond aux différents types d'erreurs présents dans l'équation 6.2. Cette liste est passée sous la forme de types génériques au solveur LMA :

```
Solver<F1,...,Fn>
```

Cela permet d'avoir accès aux types des fonctions (sans niveau d'abstraction) à la compilation. A partir de la liste de fonctions L, LMA déduit les paramètres du problème. Pour cela, on impose le formalisme suivant : les fonctions F_i doivent se présenter sous la forme de foncteurs C++, autrement dit,

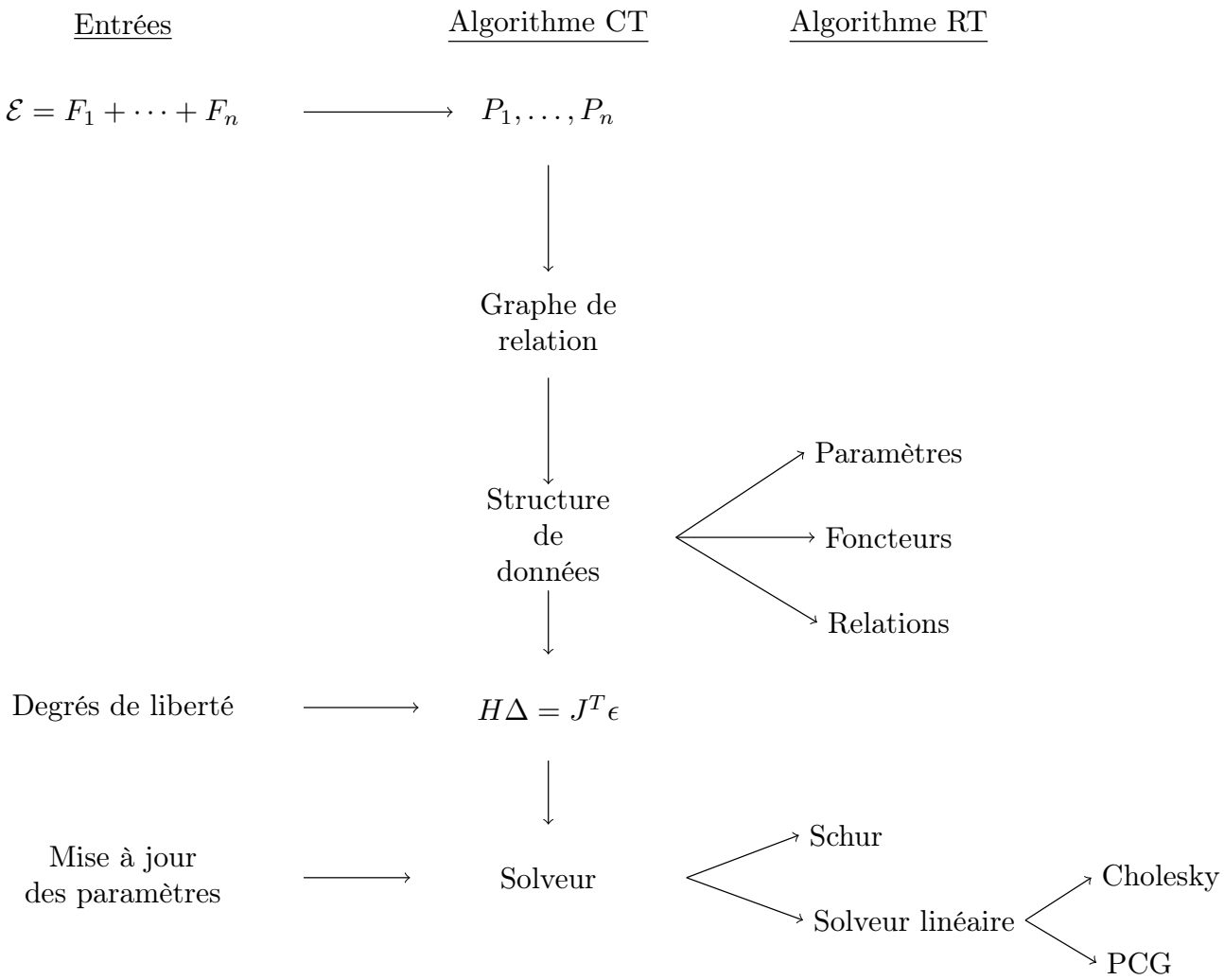


FIGURE 6.1 – Etapes de l'algorithme CT : à partir de la fonction de coût ϵ, les paramètres $P_1, P_2, ...$ sont extraits et les relations entre l'ensemble des paramètres du problème sont déduites. Puis une structure de données est construite pour contenir l'ensemble des données du problème. Ensuite, les équations normales $H\Delta = J^T\epsilon$ sont générées avec une représentation éparse efficace utilisant les *ddl.* de chaque type de paramètres. Pour finir le solveur est généré avec la méthode de résolution choisie ainsi que les méthodes de mise à jour de paramètres.

des classes objets implémentant l'opérateur parenthèses. Celui-ci renvoie un booléen correspondant à la réussite ou l'échec du calcul. De plus, la liste d'arguments de l'opérateur parenthèses est composée des paramètres à optimiser et en dernière position le résidu calculé, par exemple :

```
struct F1 {
  bool operator()(double x, double& residu) const {
    residu = x - 10.0;
    return true;
  }
};
```

où le paramètre x est optimisé pour minimiser la fonction de coût $x-10$. Si le calcul du résidu nécessite des paramètres supplémentaires à optimiser (par exemple un *double* y) ainsi que des données (comme une mesure), il est possible d'écrire :

```
struct F2 {
  double mesure;
  bool operator()(double x, double y, double& residu) const {
    residu = x + y - mesure;
    return true;
  }
};
```

De cette manière, LMA analyse l'opérateur parenthèses de chaque type de foncteurs afin de créer la liste de paramètres du problème.

6.2.2 Les paramètres

Les différents paramètres de la fonction de coût sont obtenus en parcourant la liste de fonctions d'erreurs $L = F_1, ..., F_n$, et en extrayant la liste des arguments de l'opérateur parenthèses de chacun des foncteurs. Pour cela, on définit deux méthodes qui sont exécutées pendant la compilation :

1. une méthode *extract_parameters* qui renvoie la liste d'arguments de l'opérateur parenthèses d'un foncteur,
2. l'algorithme 2 *extract_all_parameters* qui concatène et supprime les doublons des paramètres sur l'ensemble des foncteurs présentés par l'algorithme 2.

Algorithm 2: extract_all_parameters : extrait la liste de tous les paramètres du problème

input : $L_F = List < F_1, \ldots, F_n >$
output: $L_P = List < P_1, \ldots, P_m >$

$L_P = \{\}$;
foreach *element* f *from* L_F **do**
  L_P = `cat(`L_P`,extract_parameters(`f`))`;
end
L_P = `unique(`L_P`)`;

Contrairement aux autres bibliothèques de l'état de l'art, LMA n'impose aucune limitation sur le nombre d'arguments des foncteurs et donc sur le nombre de familles de paramètres à optimiser.

6.2.3 Graphe de relations entre paramètres

La fonction de coût du problème crée des liens entre les paramètres que l'on représente par un graphe de relations. Ces relations dépendent des dérivées non nulles et définissent la structure éparse

Algorithm 3: ListEdges : calcule la liste des liens entre les paramètres

input : $L_F = List\{F_1, \ldots, F_n\}$
output: $L_{Edges} = List\{Edge_0, \ldots\}$

$L_{Edges} = \{\}$;
foreach *element* F *from* L_F **do**
  $L_P =$ `extract_parameters(`F`)`;
  for $i \leftarrow 1$ **to** $\|L_P\|$ **do**
    for $j \leftarrow i + 1$ **to** $\|L_P\|$ **do**
      $L_{Edges} = L_{Edges} + \{L_P(i), L_P(j)\}$
    end
  end
end
$L_{Edges} =$ `unique(`L_{Edges}`)`;

des équations normales. Donc en fonction des liens, la jacobienne et la hessienne peuvent avoir une structure plus ou moins complexe.

Naturellement, il n'est pas possible de déterminer à la compilation les liens entre chaque instance de paramètre car cela dépend de données dynamiques (connues seulement à l'exécution). Néanmoins, il est possible de déterminer les liens entre les familles [1] de paramètres, car ces liens sont définis par les foncteurs d'erreur. En analysant chaque foncteur indépendamment, on déduit que les familles de paramètres intervenant dans un même foncteur sont liées entre elles. Ainsi, il est possible qu'il y ait des liens entre deux paramètres d'une même famille si ces deux paramètres interviennent dans un même foncteur. La procédure utilisée pour parcourir les foncteurs et déduire les liens entre paramètres est décrite par l'algorithme 3. Par exemple, la fonction de coût :

$$\mathcal{E} = \mathcal{F}_1(P_1, P_1, P_3) + \mathcal{F}_2(P_1, P_4) + \mathcal{F}_3(P_2, P_3) + \mathcal{F}_4(P_2, P_4) \tag{6.4}$$

sera implémentée dans le code source par les foncteurs F1, F2, F3, F4 :

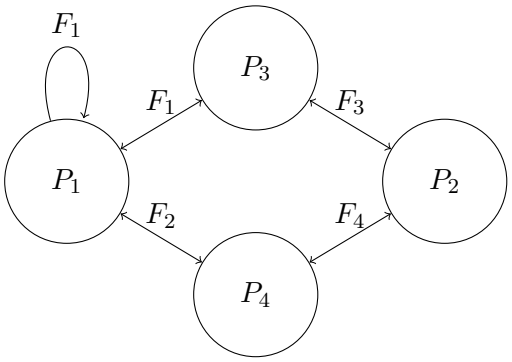
```
struct F1 {
  bool operator()(P1, P1, P3, Residual&) const {...}
};
struct F2 {
  bool operator()(P1, P4, Residual&) const {...}
};
struct F3 {
  bool operator()(P2, P3, Residual&) const {...}
};
struct F4 {
  bool operator()(P2, P4, Residual&) const {...}
};
```

Le type du solveur est défini avec les 4 foncteurs composant la fonction de coût :

```
Solver<F1,F2,F3,F4>
```

et l'algorithme 3 produira le graphe suivant :

1. Types des paramètres.



qui sera utilisé pour générer à la fois la structure de données du problème et les équations normales.

6.2.4 Structure de données

La structure de données et les algorithmes de LMA supportent un nombre variable de fonctions d'erreur et de familles de paramètres grâce à l'utilisation de conteneurs hétérogènes (tuples) générés dès le processus de compilation. Tandis que les fonctions d'erreur impactent directement la structure de la jacobienne, les relations entre les paramètres déterminent la structure de la hessienne. Ainsi, pour chaque fonction de coût, LMA déduit à partir du graphe de relations décrit précédemment une structure de données spécifique composée des éléments suivants :

— conteneurs de paramètres,
— conteneurs de foncteurs d'erreur,
— relations entre les paramètres et foncteurs d'erreur,
— relations entre paramètres,
— positions éparses dans la hessienne des couples de paramètres dépendants[2].

6.2.5 Équations normales éparses

Les équations normales correspondent au système d'équations linéaires du problème $H\Delta = J^T\epsilon$ résolu à chaque itération de l'algorithme. Il faut donc générer la jacobienne et la hessienne de manière éparse. Cela nécessite de connaître les relations entre les paramètres et les *ddl.* de chaque type de paramètre pour dimensionner les blocs de matrices.

Degrés de liberté des paramètres

Lorsque cela est nécessaire, il est possible de regrouper des paramètres dans des structures de paramètres. Par exemple, un point 3D composé de 3 scalaires est représenté par un tableau de 3 scalaires. Le fait de regrouper les paramètres permet l'utilisation d'un stockage et de calculs éparses plus efficaces. Pour calculer la jacobienne de la fonction de coût, il est nécessaire de connaître le nombre de *ddl.* associés à chaque structure de paramètres. LMA déduit automatiquement le nombre de *ddl.* pour les paramètres les plus simples comme les scalaires ou les tableaux de scalaires. Toutefois, il est possible que les paramètres soient regroupés dans des structures plus complexes qu'un tableau de scalaires. Par exemple, une pose (modélisée par une rotation et une position) contient 3×3 scalaires pour la rotation et 3 scalaires pour la position. Mais en utilisant une représentation minimale, le nombre de *ddl.* d'une pose est de 6. Pour les structures de paramètres complexes, l'utilisateur doit spécifier le nombre de *ddl.* associés. Par exemple pour la structure **Pose**, la fonction suivante doit être définie :

```
constexpr size_t ddl(Pose const &) { return 6; }
```

2. liés par le calcul de dérivée)

où **Pose** est une structure arbitrairement définie par l'utilisateur. Le mot-clé **constexpr** signifie au compilateur qu'il est possible d'évaluer le résultat de la fonction à la compilation. Ainsi, la fonction **ddl()** est utilisée, par le LMA, pour définir des types matriciels à dimensions statiques. Par exemple, le code suivant permet d'instancier une matrice de dimension 6×6 sur la pile d'appels du programme, ce qui est plus efficace que l'allocation dynamique :

```
Matrix<double,ddl(Pose{}),ddl(Pose{})> matrix66;
```

Génération des équations normales

Pour implémenter efficacement les équations normales, on utilise le graphe de relations entre les paramètres qui a été généré à partir de la liste de foncteurs 6.2.3. Chacune des relations est interprétée pour choisir une structure de données adaptée. En reprenant l'exemple de la section 6.2.3, l'analyse des paramètres de chaque foncteur permet de construire le tableau suivant :

	P_1	P_2	P_3	P_4
F_1	$\times$		$\times$	
F_2	$\times$			$\times$
F_3		$\times$	$\times$	
F_4		$\times$		$\times$

(6.5)

dont découle directement la structure de la jacobienne :

$$J = \begin{bmatrix} J_{P_1}^{F_1} & 0 & J_{P_3}^{F_1} & 0 \\ J_{P_1}^{F_2} & 0 & 0 & J_{P_4}^{F_2} \\ 0 & J_{P_2}^{F_3} & J_{P_3}^{F_3} & 0 \\ 0 & J_{P_2}^{F_4} & 0 & J_{P_4}^{F_4} \end{bmatrix} \tag{6.6}$$

Puis, on construit la hessienne à partir des relations entre paramètres du graphe :

$$H = J^T J = \left[\begin{array}{cc|cc} H_{P_1,P_1} & 0 & H_{P_1,P_3} & H_{P_1,P_4} \\ 0 & H_{P_2,P_2} & H_{P_2,P_3} & H_{P_2,P_4} \\ \hline 0 & 0 & H_{P_3,P_3} & 0 \\ 0 & 0 & 0 & H_{P_4,P_4} \end{array}\right] \tag{6.7}$$

Comme les paramètres P_1 ne sont pas liés aux paramètres P_2, la hessienne H_{P_1,P_2} sera toujours nulle, idem pour la hessienne H_{P_3,P_4}. La structure de la hessienne est alors simplifiée. On observe que les paramètres de type $P1$ sont liés entre eux. Ainsi la hessienne de P_1, notée H_{P_1,P_1} est une matrice symétrique par blocs. Tandis que les hessiennes des autres paramètres H_{P_2,P_2}, H_{P_3,P_3} et H_{P_4,P_4} sont des matrices diagonales par blocs. Les autres parties de la hessienne n'étant pas symétriques, LMA utilisera une représentation éparse par blocs classique. Le vecteur d'inconnues Δ ainsi que le vecteur d'erreurs ϵ sont implémentés avec de simples vecteurs colonne. De cette manière, LMA analyse la fonction de coût, et génère une structure spécialisée et performante avec un stockage par blocs de dimensions statiques pour la résolution des équations normales.

6.3 Résolution des équations normales

Cette section présente la résolution des équations normales effectuée à l'exécution du programme. Pour cela, l'utilisateur sélectionne un solveur linéaire ainsi que l'utilisation du complément de Schur. On présente également dans cette section la méthode du complément de Schur implicite qui est particulièrement efficace sur les problèmes de grande dimension.

	Cholesky	PCG	Schur	Schur implicite
DENSE	×			
SPARSE		×		
DENSE_SCHUR	×		×	
SPARSE_SCHUR		×	×	
IMPLICIT_SCHUR		×		×

FIGURE 6.2 – Méthodes de résolution de systèmes linéaires supportées par LMA.

Algorithm 4: Algorithme du gradient conjugué préconditionné pour la résolution du système linéaire $Ax = b$

Data: A, b, $x = \{0, \ldots, 0\}$, C le préconditionneur
Result: Le vecteur de paramètres x minimisant $\|Ax - b\|$
$r_0 = b - Ax$;
`Solve` $Cp_0 = r_0$;
$y_0 = p_0$;
for $k = 0$ **to** `Card` (x) **do**
  $\alpha_k = (r_k^T y_k)/(p_k^T A p_k)$;
  $x = x + \alpha_k p_k$;
  $r_{k+1} = r_k - \alpha_k A p_k$;
  `Solve` $Cy_{k+1} = r_{k+1}$;
  $\beta_{k+1} = (r_{k+1}^T y_{k+1})/(r_k^T y_k)$;
  $p_{k+1} = y_{k+1} + \beta_{k+1} p_k$;
  if $\|r\|/\|b\| \leq 0.9999$ **then**
    `Stop` ;
  end
end

6.3.1 Solveur linéaire

La bibliothèque LMA utilise deux méthodes pour la résolution du système d'équations linéaires : une méthode dense basée sur l'algorithme de Cholesky, et une méthode éparse par blocs avec l'algorithme du gradient conjugué préconditionné noté PCG. La méthode de Cholesky utilisée dans LMA est un appel direct de l'implémentation disponible dans la bibliothèque Eigen et ne sera donc pas présentée ici. Les algorithmes du PCG et du complément de Schur implicite ont été implémentés en utilisant une représentation éparse par blocs et sont présentés dans les sections suivantes. Les méthodes de résolution de systèmes linéaires utilisables par LMA, résumées par le tableau 6.2, sont les suivantes :

— DENSE : Cholesky dense,
— SPARSE : PCG éparse,
— DENSE_SCHUR : complément de Schur avec Cholesky dense,
— SPARSE_SCHUR : complément de Schur avec PCG éparse,
— IMPLICITE_SCHUR : complément de Schur implicite avec PCG éparse,

Une étude de complexité entre les algorithmes de Cholesky et de PCG a été réalisée par Lébraly [86].

Gradient conjugué préconditionné

Le PCG est une méthode, décrite par l'algorithme 4, particulièrement adaptée à la résolution de systèmes d'équations avec un nombre important d'inconnues [97]. Il s'agit d'un algorithme itératif

pour la résolution de systèmes linéaires de la forme $Ax = b$ où la matrice A est symétrique définie positive, x est le vecteur colonne des inconnues, et b est le vecteur colonne contenant les résidus. Le PCG minimise la fonction $g(x) = \frac{1}{2}x^T Ax - b^T x$. A chaque itération k une direction de descente p_{k+1} est choisie comme une combinaison linéaire entre la direction précédente p_k et la direction du gradient de g telles que les directions soient conjuguées : $p_{k+1}^T A p_k = 0$. L'utilisation d'une matrice de préconditionnement permet d'accélérer la convergence : plus la matrice de préconditionnement est proche de A^{-1}, plus la convergence de la méthode est rapide. Agarwal *et al.* présentent plusieurs stratégies pour générer une matrice de préconditionnement efficace [98, 99], toutefois cela n'a d'intérêt que si le temps nécessaire au calcul du préconditionneur est faible par rapport au temps total de la résolution. En pratique, nous utilisons le préconditionneur de Jacobi [100] correspondant aux blocs diagonaux inversés de la matrice de Schur.

L'algorithme du PCG est considéré plus rapide que les autres méthodes car en pratique, peu d'itérations sont effectuées. La solution obtenue est donc approximative. Toutefois, on sait que l'algorithme de Levenberg-Marquardt approxime la matrice hessienne par $J^T J$. Ainsi, il n'est pas pertinent de rechercher la solution exacte de l'itération courante (car il s'agit de la solution d'un problème approximé). L'intérêt est de passer rapidement à l'itération suivante afin de mettre à jour les paramètres et les directions de descente pour converger rapidement vers la solution. Il a été montré par Jeong *et al.* [100] que le PCG est la méthode offrant le meilleur compromis entre temps de calcul et consommation mémoire pour des méthodes d'AF impliquant plusieurs centaines de poses.

Complément de Schur Implicite

On a vu dans la section 3.15 que l'utilisation du complément de Schur consiste à calculer (dans un premier temps) Δ_C en résolvant le système d'équations linéaires de la forme $Ax = b$ suivant :

$$(U - WV^{-1}W^T)\Delta_C = \epsilon_C - WV^{-1}\epsilon_P \tag{6.8}$$

Toutefois, le calcul de la matrice de Schur $S = (U - WV^{-1}W^T)$ nécessite d'effectuer un produit matrice par matrice : $(WV^{-1})W^T$. Or, la complexité d'un produit matriciel est de $\theta(N^3)$. Ainsi, avec un nombre de paramètres important, le calcul de S devient extrêmement coûteux en temps de calcul et en consommation mémoire. Dans les cas où N est grand, il est avantageux d'utiliser la méthode du complément de Schur implicite qui, combinée avec l'algorithme du PCG, permet de résoudre les équations normales sans calculer explicitement S. Pour cela, il faut remarquer que, dans le PCG décrit par l'algorithme 4, la matrice de Schur (notée A dans l'algorithme du PCG) n'est utilisée que pour faire un produit matrice par vecteur noté Ap_k. En développant Ap_k, que l'on note ici par le vecteur $S_p = S \times p$ avec S la matrice de Schur et p un vecteur colonne, on obtient :

$$S_p = (U - WV^{-1}W^T)p \tag{6.9}$$

En développant l'équation, le calcul de S_p se ramène à la soustraction de deux vecteurs colonne sans qu'aucun produit matrice par matrice n'ait été effectué :

$$S_p = \underbrace{\underbrace{Up}_{v0=Up} - W\,\underbrace{V^{-1}\,\underbrace{W^T p}_{v1=W^T p}}_{v2=V^{-1}v1}}_{v3=Wv2} \qquad \underbrace{\qquad\qquad}_{Sp=v0-v3} \tag{6.10}$$

La complexité du calcul de S_p dépend alors des multiples produits matrice par vecteur, elle est donc de l'ordre de $\theta(N^2)$, tandis que la complexité du calcul de S est de $\theta(N^3)$. L'utilisation du complément de Schur implicite permet donc en théorie une réduction de la complexité du problème. Toutefois,

pour résoudre les équations normales avec le complément de Schur explicite, S est calculée une seule fois. Tandis que S_p est calculée à chaque itération du PCG avec la méthode implicite. C'est pourquoi, cette méthode montre un intérêt uniquement si le nombre d'itérations du PCG est relativement faible et si le nombre de paramètres du problème est lui relativement important. La convergence de cette méthode a été démontrée par Wright *et al.* [101]. En pratique, on utilise le complément de Schur implicite uniquement sur les problèmes avec un grand nombre de paramètres.

Mise à jour des paramètres

A chaque itération de Levenberg-Marquardt, il est nécessaire de mettre à jour les paramètres en utilisant le résultat Δ issu de la résolution des équations normales. Pour chaque type de paramètre, la dimension de l'incrément correspond aux *ddl.* du type de paramètre. Le plus souvent, la mise à jour des paramètres est triviale. Par exemple, à un point 3D correspond 3 *ddl.* donc la mise à jour de ce type de paramètre consiste à incrémenter les coordonnées du point 3D avec les 3 incréments calculés par le solveur. Mais il existe plusieurs manières de mettre à jour une matrice de rotation à partir de 3 incréments, par exemple avec les angles d'Euler, les quaternions ou la paramétrisation axes angles. Ainsi, les méthodes de mise à jour des paramètres doivent être définies pour chaque type, par exemple pour le type **Pose** :

```
void apply_increment(Pose& pose, double increment[6])
{
  ...
}
```

Pour le calcul de dérivée numérique, il est possible de spécialiser l'application d'incréments infinitésimaux sur certains degrés de liberté, déterminés par l'indice I, en particulier :

```
template<int I>
void apply_small_increment(Pose& pose, double increment, Int<I>)
{
  ...// increment sur le parametre I
}
```

Cette surcharge optionnelle rend possible l'application de rotations infinitésimales utilisées avec la représentation exponentielle locale (*cf. 1.4*) qui sont plus rapides à calculer.

6.4 Fonctionnalités

On présente dans cette section différentes fonctionnalités de la bibliothèque :

— les calculs effectués lors de la résolution des équations normales sont génériques sur le type matriciel. Dans la version actuelle, deux bibliothèques matricielles sont supportées : Eigen [3] et Toon [4],

— le calcul des dérivées peut être fait de manière numérique, analytique (à définir par l'utilisateur) ou en utilisant une méthode de calcul de dérivées automatique,

— un support pour les M-estimateurs (présentés brièvement dans la section 3.2.2) est également inclu pour simplifier la gestion des valeurs aberrantes mais ne sera pas présenté ici.

3. eigen.tuxfamily.org/
4. github.com/edrosten/TooN

6.4.1 Bibliothèques matricielles

LMA génère une structure éparse par blocs correspondant aux équations normales du problème à résoudre. L'intérêt est ici d'utiliser des bibliothèques matricielles efficaces pour les calculs algébriques basiques sur des matrices dont la taille dépend du nombre de *ddl.* de blocs de paramètres.

Les calculs effectués à l'échelle des blocs sont relativement simples. Il s'agit des opérations matricielles suivantes : $+$, $-$, $\times$, inversion, transposition. De nombreux progrès ont récemment été faits dans l'implémentation de bibliothèques matricielles performantes basées sur l'utilisation d'expressions *template* et d'optimisations propres à l'architecture matérielle : Eigen, TooN, NT2, Blaze, Armadillo, ...

Le choix de la bibliothèque matricielle pour les calculs sur des matrices de petites tailles est déterminant pour les performances de LMA. Étant donné que ces bibliothèques sont amenées à évoluer, et que la notion de « meilleure » bibliothèque est subjective voire liée au problème à résoudre, LMA est conçue pour être générique sur le type de bibliothèque matricielle. Ainsi, LMA est en théorie capable d'utiliser n'importe quelle bibliothèque matricielle pour résoudre les calculs à l'échelle des blocs. Cela est possible grâce à la programmation générique. Le choix de la bibliothèque matricielle est effectué à la compilation en fonction d'un tag (l'utilisateur choisit le tag correspondant à la bibliothèque à utiliser), ici Eigen_ ou TooN_. La valeur oriente la sélection du type de bloc par spécialisation de *template* :

```
template<int I, int J, class Float, class Tag>
struct Bloc;

template<int I, int J>
struct Bloc<I,J,Float,Eigen_> : Eigen::Matrix<Float,I,J> {};

template<int I, int J>
struct Bloc<I,J,Float,TooN_> : TooN::Matrix<I,J,Float> {};

template<int I>
struct Bloc<I,1,Float,TooN_> : TooN::Vector<I,J,Float> {};
```

La bibliothèque TooN utilise une structure différente pour les matrices et les vecteurs (ou matrice colonne). Toutefois, la spécialisation du type bloc pour une unique colonne permet de choisir la structure adéquate. Une approche générique est requise pour implémenter les opérations. Prenons le calcul suivant :

```
a = b * c + d;
```

Les bibliothèques Eigen et TooN interprètent ce calcul de manière identique, donc cette expression est générique. Mais un calcul faisant intervenir une transposition prend deux écritures différentes :

```
// Eigen
a = b * c + d.transpose();
// TooN
a = b * c + d.T();
```

Dans ce cas, l'expression ne compilera pas avec les deux bibliothèques. On définit alors un langage minimal utilisé au sein de LMA pour unifier l'écriture des calculs. Ainsi, la transposition devient une fonction globale nommée *transpose* et définie à la fois pour les types Eigen et TooN :

```
// Eigen tranpose
template<class Derived>
auto transpose(const Eigen::DenseBase<Derived>& m) { return m.transpose(); }
```

```
// TooN transpose
template<class Float, int I, int J>
auto transpose(const TooN::Matrix<I,J,Float>& m) { return m.T(); }

// Generic
a = b * c + transpose(d);
```

L'utilisation d'un type de retour automatiquement déduit (avec le mot clé **auto**), permet aux moteurs d'expressions propres à chacune des bibliothèques d'effectuer des optimisations sur les expressions générées. En pratique, on observe des différences significatives de performances en fonction de la bibliothèque matricielle utilisée. Un comparatif est proposé dans les expérimentations 7.1.

6.4.2 Calcul de dérivées

Cette section présente les 4 méthodes de calcul de dérivées utilisables dans LMA pour évaluer la jacobienne de la fonction de coût : dérivées numérique et centrée ainsi qu'automatique et analytique.

Dérivée numérique

Le calcul de dérivée d'une fonction f en un point x consiste à calculer la variation de f selon un déplacement infinitésimal de x. Pour cela, la dérivée numérique est approximée par des taux d'accroissement

— dérivée numérique centrée :

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h} \tag{6.11}$$

— dérivée numérique à gauche :

$$f'(x) \approx \frac{f(x+h) - f(x)}{h} \tag{6.12}$$

En pratique, on utilise comme valeur de pas $h = \sqrt{\epsilon}$ avec ϵ la précision de la machine. Autrement dit, h correspond au plus petit nombre α tel que, numériquement, $1 + \alpha \neq 1$. La valeur théorique de l'erreur liée aux approximations ainsi que les dérivées d'ordres supérieurs à 1 sont détaillées par Lapresté dans [102].

Dérivée automatique

La différentiation automatique est un ensemble de techniques d'évaluation des dérivées en un point d'une fonction écrite dans un programme informatique en utilisant un langage de haut niveau. Pour cela, le programme évalue de manière automatique la fonction en appliquant les règles de dérivation des fonctions composées à appliquer aux valeurs numériques. Contrairement aux méthodes numériques, les méthodes automatiques ne font aucune approximation, donc le résultat est exact. Toutefois, ces techniques sont utilisables uniquement si la fonction à dériver est écrite avec des opérateurs et des fonctions mathématiques dont la forme des dérivées est connue. De nombreuses méthodes et programmes pour la différentiation automatique existent et sont répertoriés sur le site autodiff.org.

La méthode de dérivée automatique utilisée dans LMA se base sur l'algèbre des nombres duaux introduit par Clifford [103]. Une implémentation en C++ est présentée par Piponi dans [104]. Les nombres duaux se composent d'une partie réelle et d'une partie infinitésimale ϵ telle que $\epsilon^2 = 0$ et peuvent s'exprimer sous la forme $x + \epsilon$. Ainsi, en évaluant la fonction suivante :

$$f(x) = x^2 \tag{6.13}$$

avec le nombre dual $10 + \epsilon$:

$$f(10 + \epsilon) = (10 + \epsilon)^2 \tag{6.14}$$

on obtient :

$$f(10 + \epsilon) = \underbrace{10^2}_{real} + \underbrace{2 \times 10 \times \epsilon}_{infinitesimal} + \underbrace{\epsilon^2}_{=0} \tag{6.15}$$

Avec $\epsilon^2 = 0$, la partie réelle de $f(10 + \epsilon)$ est égale à $f(10)$ donc à 100, et la dérivée de la partie infinitésimale $20 \times \epsilon$ est de 20 :

$$f(10 + \epsilon) = \underbrace{100}_{f(10)} + \underbrace{20 \times \epsilon}_{f'(10)} \tag{6.16}$$

L'implémentation proposée par Piponi consiste à définir une structure duale contenant à la fois la partie réelle et la partie infinitésimale. Puis il redéfinit les opérateurs usuels $+, -, \times, /, cos, exp \ldots$ ce qui prend la forme suivante :

```
struct Duale {
  Real real;
  Infinite infinite;
};

Duale operator+(Duale a, Duale b) {
  return Duale{a.real+b.real,a.infinite+b.infinite}; }
...
```

Dans l'approche de Piponi [104], chaque opérateur effectue un calcul et crée un nouveau nombre dual contenant le résultat. Toutefois, ce type d'approche génère beaucoup de variables temporaires et ne permet pas d'évaluer le résultat d'une manière plus globale afin d'améliorer les performances. Il s'agit de l'implémentation utilisée par Ceres et g2o pour les calculs de dérivées automatiques.

Nous proposons ici une amélioration significative des performances d'exécutionRéduction de 40% du temps de calcul pour calculer la dérivée d'une fonction d'erreur de reprojection. en utilisant les récentes fonctionnalités du C++ à savoir la déduction automatique de type combiné à l'utilisation d'un moteur d'expressionsNotre . Un moteur d'expressions (notion présentée dans [43, 105] ainsi que dans la section 1.12.4) permet de construire un arbre syntaxique de l'expression totale de la fonction à dériver. Il est alors possible d'effectuer des simplifications et des optimisations de cette expression pendant le processus de compilation comme par exemple en créant un minimum de variables duales temporaires. Bien que simplifiée par des bibliothèques comme Boost.Proto, l'écriture d'un moteur d'expressions reste un exercice très technique. Notre idée est d'implémenter un moteur d'expressions pour le type « dual » en utilisant un moteur d'expressions existant : celui de la bibliothèque matricielle utilisée par LMA (au choix de l'utilisateur Eigen ou TooN). Ainsi dans notre implémentation, les parties réelle et infinitésimale du type dual sont un type natif de la bibliothèque matricielle utilisée. Les opérateurs usuelles ont été redéfinis pour ce nouveau type duale. Ces opérateurs effectuent les calculs de parties réelle et infinitésimale afin de générer des expressions qui sont optimisées et évaluées par la bibliothèque matricielle elle-même. Pour plus de détails sur l'implémentation, le lecteur intéressé pourra se référer directement au code source de LMA [5].

Dérivée analytique

LMA peut utiliser la formule analytique de la dérivée, lorsque celle-ci est connue. Naturellement, cette approche est la plus efficace en matière de temps de calcul et de précision.

5. lma/numeric/ad/ct/adct.hpp

6.5 Interface utilisateur

L'interface utilisateur de LMA est un langage intermédiaire, compréhensible à la fois pour le développeur et par LMA, pour modéliser un problème d'optimisation. Le méta-programme utilise ce modèle pour générer le solveur adéquat. L'interface de LMA est conçue pour être simple tout en permettant l'analyse du code par le méta-programme. Considérons un foncteur d'erreur impliquant 3 types différents de paramètres **X**,**Y** et **Z** :

```
struct Error {
  bool operator()(X x, Y y, Z z, double& r) const {...}
};
```

La création du solveur doit permettre la génération de la structure de données à la compilation, donc le type du foncteur d'erreur doit être passé en argument *template* :

```
Solver<Error> solver;
```

Puis, les données sont envoyées au solveur en utilisant une unique fonction **add** qui prend en arguments le foncteur d'erreur (correspondant à un terme de la fonction de coût) et les paramètres associés à cette erreur :

```
X x; Y y; Z z;
solver.add(Error(),&x,&y,&z);
```

Avec cette unique ligne, LMA va mettre à jour le graphe de paramètres en liant les types de x, y et z entre eux. En se basant sur l'adresse mémoire des arguments, LMA identifie s'il s'agit de nouveaux paramètres ou de paramètres déjà présents dans le graphe de paramètres. De plus, LMA associe l'instance de la fonction d'erreur **Error()** aux arguments pour pouvoir évaluer l'erreur avec les arguments correspondants. La résolution du problème se fait par un appel à la fonction **solve** avec la méthode de résolution choisie :

```
solver.solve(DENSE);// EPARSE, DENSE_SCHUR ...
```

Le code 6.1 correspond à un exemple d'utilisation de LMA pour trouver le minimum de la fonction de Rosenbrock [6]. La fonction d'erreur est de type **Rosenbrock**, le paramètre **r** est le résidu qui sera utilisé par le solveur. Les deux paramètres à optimiser **x** et **y** sont initialisés à la valeur -1. Un solveur est généré à partir du type de la fonction d'erreur. Les paramètres ainsi que la fonction d'erreur sont mis en entrée du solveur. Puis es paramètres sont optimisés en utilisant une méthode de résolution dense.

Code 6.1 – Résolution de la fonction de Rosenbrock avec LMA.

```
#include <lma/lma.hpp>
int main() {
  struct Rosenbrock{
    bool operator()(double x, double y, double& r) const{
      r = (1.-x)*(1.-x)+100.*(y-x*x)*(y-x*x);
      return true;
    }};

  double x(-1),y(-1);
  lma::Solver<Rosenbrock> solver;
  solver.add(Rosenbrock(),&x,&y).solve(lma::DENSE);
}
```

6. $Rosenbrock(x, y) = (1 - x)^2 + 100(y - x^2)^2$

Chapitre 7

Résultats

Les expérimentations présentées ici comparent les performances d'exécution ainsi que la précision de la bibliothèque LMA avec les bibliothèques g2o et Ceres sur divers problèmes d'optimisation. Les temps d'exécution sont exprimés en secondes et la précision relative entre les 3 bibliothèques est évaluée avec le RMS (Root Mean Square) final du problème optimisé. Pour chaque test, les 3 bibliothèques sont configurées de manière identique en termes de méthode de résolution, de nombre de *thread* (1 seul) et de nombre d'itérations (20 itérations de Levenberg-Marquardt et 20 itérations de PCG lorsque celui-ci est utilisé). Les algorithmes testés sont les différentes méthodes de résolution de systèmes linéaires présentées dans le tableau 6.2. ainsi que les méthodes de dérivées présentées dans la section 6.4.2 qui seront notées :

- NDC : dérivée numérique centrée,
- NDG : dérivée numérique à gauche,
- AD : dérivée automatique,
- AN : dérivée analytique.

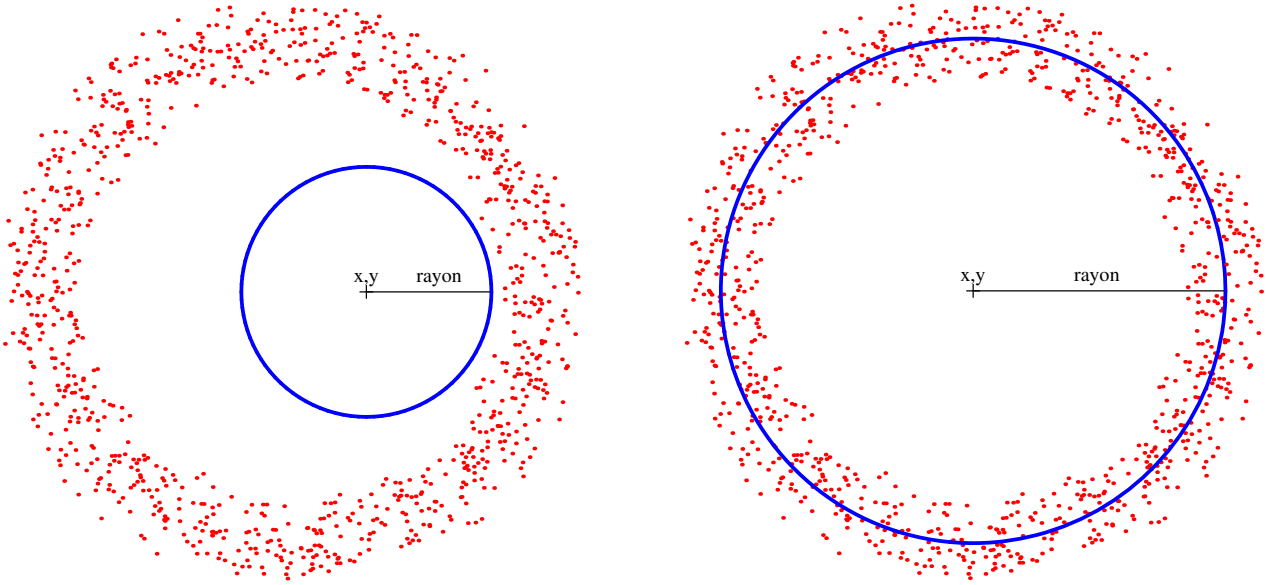
Pour effectuer cette série d'expérimentations, on a sélectionné 3 problèmes d'optimisation de nature et de complexités algorithmiques différentes :

- optimisation des paramètres d'un cercle $(x, y, rayon)$ en minimisant la distance entre des points 2D et le cercle (seuls les paramètres du cercle sont optimisés) ce qui est illustré par la figure 7.1,
- optimisation des paramètres d'une sphère $(x, y, z, rayon)$ en minimisant la distance entre des points 3D et la sphère ce qui est illustré par la figure 7.2 (optimisation à la fois de la sphère et des points 3D),
- 35 problèmes d'AF issus du jeu de données public BAL dont la figure 7.3 est un exemple et disponible en ligne [1].

Les caractéristiques (nombre de paramètres, nombre d'observations, dimensions et nombre de blocs de paramètres) des différents problèmes traités sont présentées dans le tableau en annexe 11.4. Par exemple, le problème Bal_0 est un problème à 23769 paramètres et 63686 résidus. Les paramètres sont composés de 49 poses ($\#blocs_1$), et 7776 points 3D ($\#blocs_2$). On compte 9 *ddl.* par pose et 3 *ddl.* par point 3D.

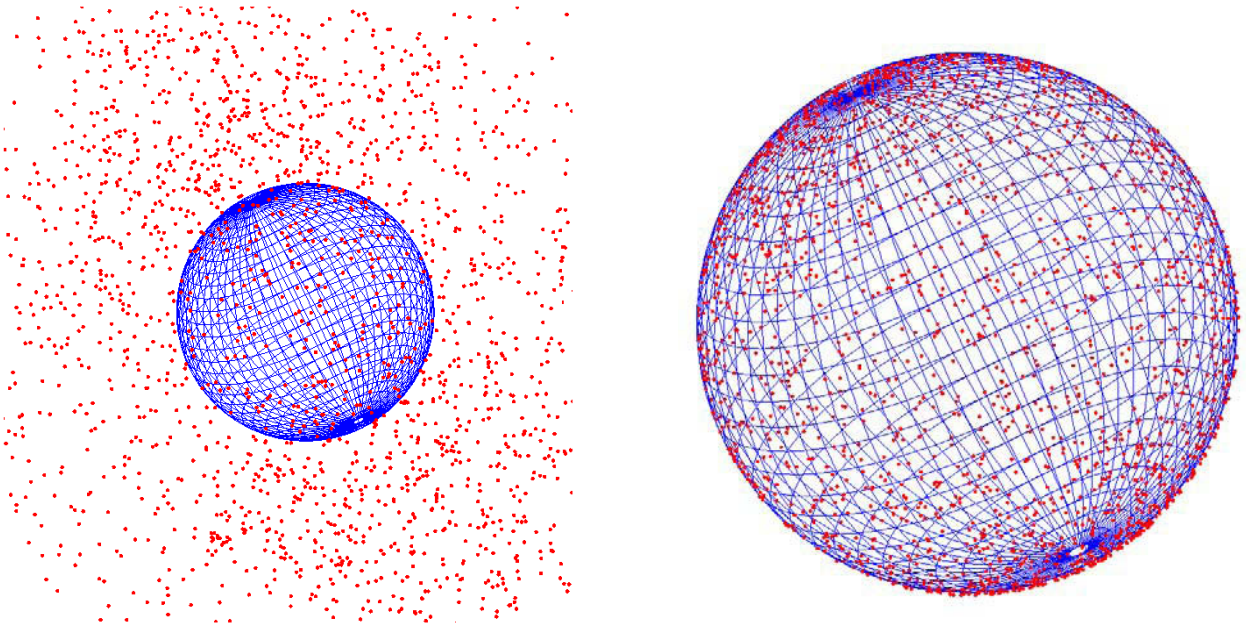
Avec 3 paramètres à optimiser, le problème du cercle est résolu en utilisant la méthode DENSE. Les bibliothèques g2o, Ceres et LMA sont configurées pour utiliser l'algorithme de Cholesky de la bibliothèque Eigen lors de la résolution des équations normales. Le problème de la sphère permet d'utiliser le complément de Schur avec une matrice U très petite (4×4) et une matrice V grande (594405×594405). Pour résoudre ce problème, la méthode DENSE_SCHUR est utilisée. Pour les problèmes d'AF notés Bal_i, il est possible d'utiliser à la fois les méthodes SPARSE, DENSE_SCHUR, SPARSE_SCHUR et IMPLICIT_SCHUR. Le temps d'exécution des méthodes DENSE_SCHUR et

1. http://grail.cs.washington.edu/projects/bal/



(a) Avant optimisation du cercle. (b) Après optimisation du cercle.

FIGURE 7.1 – Problème d'optimisation d'un cercle afin de minimiser la somme des distances entre les points 2D et le cercle.



(a) Avant optimisation des paramètres. (b) Après optimisation des paramètres.

FIGURE 7.2 – Problème d'optimisation d'une sphère et d'un ensemble de points 3D afin de minimiser la somme des distances entre les points 3D et la sphère.

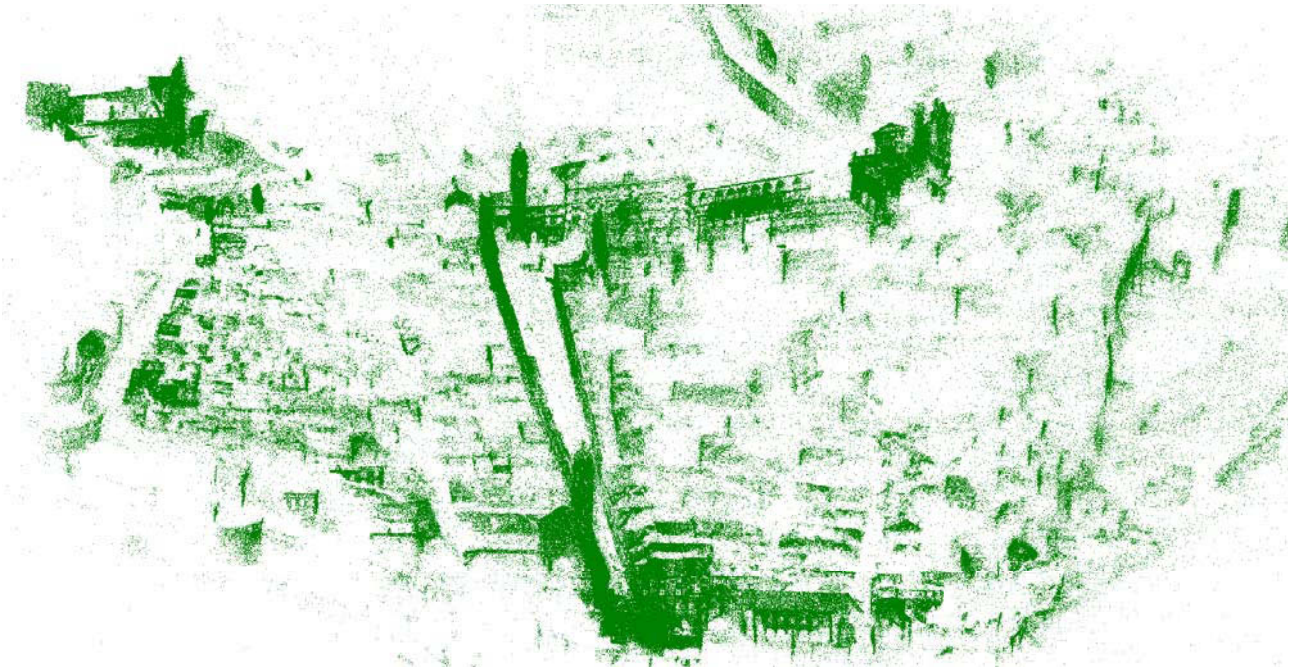


FIGURE 7.3 – Nuage de points 3D issus du jeu de données BAL_{34} après un ajustement de faisceaux.

SPARSE_SCHUR pour les problèmes à plusieurs milliers de poses est important, c'est pourquoi on limitera l'utilisation de ces deux méthodes aux problèmes impliquant moins de 1500 poses.

Certains tests n'ont pas pu être réalisés : la bibliothèque g2o n'implémente pas la méthode du complément de Schur implicite ; g2o ne permet pas non plus d'utiliser la méthode du PCG éparse sans effectuer le complément de Schur (ce qui correspond au test de la méthode SPARSE) ; de plus, g2o ne permet pas de mesurer le temps nécessaire au calcul des dérivées et ne sera donc pas utilisée pour le comparatif sur les dérivées.

7.1 Eigen ou TooN

On a vu précédemment que LMA peut supporter différentes bibliothèques matricielles pour effectuer les calculs au niveau des blocs. On compare ici l'utilisation des bibliothèques Eigen et Toon sur différents problèmes d'optimisation. L'erreur finale est identique pour les deux approches car le choix de la bibliothèque matricielle n'a pas d'influence sur la précision des calculs. D'après le tableau 7.1 on remarque que l'utilisation de TooN dans le solveur de LMA offre un meilleur temps de calcul sur les problèmes *Bal* tandis que Eigen est plus performant sur les deux premiers problèmes. Dans l'ensemble des expérimentations restantes, on choisit d'utiliser uniquement la bibliothèque TooN.

7.2 Comparaison performances générales

Le tableau 7.2 montre le temps nécessaire à l'évaluation de la fonction de coût ainsi que la résolution des équations en fonction des 3 bibliothèques. L'évaluation de la fonction de coût est en moyenne 10 fois plus rapide avec LMA qu'avec Ceres et 5 fois plus rapide que g2o. La fonction de coût étant identique pour les 3 bibliothèques, cette différence est uniquement due à l'absence de polymorphisme dans LMA. Pour la résolution des équations normales, la méthode DENSE_SCHUR est utilisée (basée sur la méthode de Cholesky dense de Eigen pour les 3 bibliothèques). Ainsi la différence de temps

Solveur	LMA_{Eigen}	LMA_{TooN}
Cercle	**0.025476**	0.02571
Sphere	**1.03434**	1.20392
Bal_0	1.34291	**0.868372**
Bal_1	1.2378	**0.827956**
Bal_2	2.1323	**1.35982**
Bal_3	3.01913	**1.98887**
Bal_4	15.5753	**9.17691**
Bal_5	23.5303	**12.2257**
Bal_6	16.2543	**10.2839**
Bal_7	17.7999	**11.0437**
Bal_8	21.3785	**13.5767**
Bal_9	121.027	**52.5857**
Bal_{10}	36.5077	**23.1629**
Bal_{11}	50.1707	**31.4611**
Bal_{12}	78.9197	**47.0546**
Bal_{13}	121.935	**62.1896**
Bal_{14}	199.057	**95.7213**

TABLE 7.1 – Temps d'exécution en secondes de LMA en utilisant les bibliothèques Eigen et TooN pour effectuer les calculs au niveau des blocs de matrice avec la méthode DENSE pour le problème du cercle et DENSE_SCHUR pour les autres problèmes.

d'exécution pour la résolution des équations normales est due à l'évaluation du complément de Schur. Pour résoudre les équations normales, LMA est en moyenne 1.7 fois plus rapide que CERES et 3 fois plus rapide que g2o.

Le tableau 7.3 montre le temps nécessaire à l'évaluation de la jacobienne en fonction des 4 méthodes de calcul de dérivées de CERES et LMA. On note que LMA est 2 à 3 fois plus rapide que Ceres (différence due encore une fois à l'absence de polymorphisme dans LMA). On remarque que la différence est moins importante avec la méthode automatique AD car celle-ci ne fait qu'un seul appel aux foncteurs pour calculer les dérivées tandis que les méthodes NDC et NDG font autant d'appels qu'il y a de paramètres à dériver. Or, plus il y a d'appels aux foncteurs, plus Ceres est pénalisée par l'utilisation du polymorphisme.

Les tableaux 11.5,11.6,11.7 et 11.8 (déplacés en annexe), montrent les temps de calculs ainsi que le RMS final des 3 bibliothèques en fonction des différents algorithmes de résolution. Un échantillon de ce comparatif exhaustif est donné par le tableau 7.4 sur 3 problèmes.

Bien que LMA soit tout le temps plus rapide que g2o et Ceres, on remarque que Ceres offre souvent un résultat plus précis. g2o et LMA implémentent l'algorithme de Levenberg-Marquardt avec une mise à jour classique du paramètre d'amortissement (rappelée par l'algorithme 1). Tandis que Ceres utilise un algorithme de région de confiance qui est plus robuste et converge plus efficacement. De plus, on remarque que LMA est moins précis avec la méthode du complément de Schur implicite IMPLICIT_SCHUR dans l'expérimentation 11.8 qu'avec l'équivalent explicite SPARSE_SCHUR dans l'expérimentation 11.6. Cela est dû à la matrice de pré-conditionnement du PCG qui est différente dans ces deux cas. Dans le cas explicite, on utilise le préconditionneur de Jacobi sur la matrice de Schur S (ce qui correspond à l'inverse de la diagonale de S). Dans le cas implicite, la matrice S n'est pas explicitement calculée, ainsi le préconditionneur utilisé est le Jacobi de la matrice U qui est moins efficace. C'est pourquoi la méthode du complément de Schur implicite de LMA, avec un préconditionneur et l'algorithme de Levenberg-Marquardt classique, est plus rapide mais moins précise que l'implémentation de Ceres.

	Coût			Equation normales		
Solveur	g2o	Ceres	LMA	g2o	Ceres	LMA
Cercle	0.011	0.023	**0.001**	0.000	0.006	**0.000**
Sphere	0.161	0.321	**0.022**	0.760	5.506	**0.381**
Bal_0	0.049	0.095	**0.009**	1.290	0.631	**0.369**
Bal_1	0.055	0.113	**0.011**	0.893	0.512	**0.270**
Bal_2	0.074	0.144	**0.014**	2.438	1.065	**0.663**
Bal_3	0.127	0.267	**0.025**	2.219	1.142	**0.721**
Bal_4	0.413	0.740	**0.062**	17.656	8.022	**5.826**
Bal_5	0.416	0.819	**0.069**	23.102	11.012	**8.495**
Bal_6	0.507	0.989	**0.088**	15.949	7.658	**5.344**
Bal_7	0.555	1.118	**0.107**	17.094	8.585	**5.791**
Bal_8	0.660	1.277	**0.118**	20.527	9.852	**6.990**
Bal_9	0.923	1.788	**0.167**	203.473	64.646	**41.455**
Bal_{10}	0.966	1.911	**0.175**	39.021	17.978	**12.754**
Bal_{11}	1.240	2.534	**0.231**	56.826	26.158	**17.839**
Bal_{12}	1.858	3.634	**0.337**	97.452	44.676	**28.585**
Bal_{13}	2.102	4.290	**0.386**	150.563	60.938	**39.500**
Bal_{14}	3.048	5.840	**0.524**	260.214	109.214	**63.830**

TABLE 7.2 – Temps d'exécution en secondes de g2o, Ceres et LMA pour l'évaluation de la fonction de coût et la résolution des équations normales avec la méthode DENSE_SCHUR.

	CERES				LMA			
Solveur	NDC	NDG	AD	AN	NDC	NDG	AD	AN
Cercle	0.0888613	0.0870418	0.0869722	0.0795794	**0.013**	**0.012**	**0.011**	**0.009**
Sphere	0.583234	0.515046	0.471171	0.382722	**0.378**	**0.298**	**0.244**	**0.252**
Bal_0	1.1149	0.682111	0.635908		**0.342**	**0.269**	**0.171**	
Bal_1	0.724842	0.599849	0.450268		**0.392**	**0.305**	**0.184**	
Bal_2	1.4245	0.966694	0.946759		**0.569**	**0.450**	**0.303**	
Bal_3	1.39807	1.04186	0.738149		**0.791**	**0.614**	**0.341**	
Bal_4	6.47972	3.95057	3.49642		**1.981**	**1.566**	**1.010**	
Bal_5	8.38499	5.13671	4.52521		**2.049**	**1.377**	**1.051**	
Bal_6	5.67287	2.8549	2.8569		**2.868**	**2.138**	**1.164**	
Bal_7	9.99462	6.12987	5.79125		**4.332**	**3.445**	**2.205**	
Bal_8	11.933	7.35916	6.90294		**4.942**	**3.765**	**2.550**	
Bal_9	13.2311	8.0841	7.39598		**5.663**	**4.627**	**3.006**	
Bal_{10}	20.8507	12.7629	11.2705		**7.532**	**6.129**	**4.165**	
Bal_{11}	22.3794	14.5169	10.4473		**10.443**	**8.006**	**5.461**	
Bal_{12}	28.4472	14.0305	16.2075		**8.829**	**7.127**	**4.598**	
Bal_{13}	44.1571	27.1934	25.0941		**18.114**	**13.867**	**9.386**	
Bal_{14}	47.5712	29.3702	27.2145		**15.028**	**12.247**	**7.932**	

TABLE 7.3 – Temps d'exécution en secondes de Ceres et LMA pour le calcul de la jacobienne en fonction des 4 méthodes de dérivées NDC, NDG, AD et AN (la méthode DENSE_SCHUR est utilisée pour la résolution des équations normales).

	Temps			RMS		
Solver	g2o	Ceres	LMA	g2o	Ceres	LMA
DENSE Cercle	0.199	0.217	**0.025**	0.024	0.024	**0.024**
DENSE_SCHUR Sphère	3.398	8.048	**1.209**	0.000	0.000	0.000
DENSE_SCHUR Bal_0	3.570	1.978	**0.777**	0.653	**0.647**	0.650
SPARSE Bal_0		2.642	**0.984**		**0.647**	0.647
SPARSE_SCHUR Bal_0	3.331	1.284	**0.835**	**0.647**	0.649	0.647
IMPLICIT_SCHUR Bal_0		2.237	**0.978**		**0.647**	0.648
IMPLICIT_SCHUR Bal_{34}		828.191	**363.757**		**0.750**	0.804

TABLE 7.4 – Résumé du comparatif disponible en annexe : 11.6.

7.3 Conclusion

Ces expérimentations montrent que la bibliothèque LMA offre de meilleures performances en matières de temps d'exécution que les bibliothèques g2o et Ceres. Cette différence est le résultat des optimisations de code effectuées pendant la phase de compilation du solveur de LMA. Les tests ont été réalisés sur 3 types de problèmes afin de comparer 5 méthodes de résolution de système linéaire et 4 méthodes de dérivées. Toutefois, la bibliothèque Ceres offre des résultats plus précis grâce à l'utilisation de la méthode par Région de confiance. Il aurait été possible d'inclure d'autres bibliothèques dans ce comparatif, comme GTSAM ou pba (présentées en début de partie 5.1). Toutefois, GTSAM offre de moins bonnes performances que g2o sur les jeux de données BAL. Et pba est exclusivement conçue pour résoudre les problèmes d'AF issus des données BAL. Sur ces jeux de données, pba offre de meilleures performances (en temps de calcul) que LMA de l'ordre de 25% à configuration égale (nombre de *thread*, précision numérique...). Toutefois, il n'est pas possible d'utiliser pba pour résoudre d'autres problèmes (où la fonction de coût est différente). C'est pourquoi, ces deux bibliothèques n'ont pas été retenues pour les expérimentations.

Conclusion

Le tableau 7.5 présente un résumé des fonctionnalités et méthodes de résolution des bibliothèques g2o, Ceres et LMA. Les expérimentations ont montré que la bibliothèque LMA est plus performante en temps d'exécution que g2o et Ceres. Toutefois, l'algorithme de Levenberg-Marquardt utilisé donne une solution moins précise que l'algorithme à régions de confiance de Ceres. De plus, le processus de génération automatique du solveur de LMA présente des inconvénients :

— un temps de compilation relativement important (de l'ordre de 5 secondes pour un problème à 2 familles de paramètres, et 20 secondes pour un problème à 4 familles de paramètres),
— il n'est possible de définir qu'une seule paramétrisation par type de paramètre,
— l'implémentation est spécialisée pour les problèmes éparses,
— une modification de la fonction de coût nécessite la re-compilation du code.

De futurs travaux sur la bibliothèque LMA sont prévus pour en faire un outil plus complet pour l'optimisation de problèmes non-linéaires. Par exemple, implémenter l'algorithme de Cholesky éparse par blocs [106] pour améliorer les performances sur les problèmes de petites et moyennes dimensions. De plus, l'implémentation de l'algorithme de région de confiance [107] améliora la vitesse de convergence.

Fonctionnalités	g2o	Ceres	LMA
Générique sur le type de matrice			×
Générique sur le type de paramètre	×		×
# Types de paramètres illimités			×
Pas d'abstraction sur les foncteurs			×
Pas d'abstraction sur les paramètres		×	×
Blocs statiques			×
Interface simple		×	×
Solveur	g2o	Ceres	LMA
Dérivée automatique	×	×	×
Fonction robuste	×	×	×
Schur dense/éparse	×	×	×
Éparse/Schur implicite		×	×
Région de confiance		×	

TABLE 7.5 – Fonctionnalités et méthodes de résolution des bibliothèques g2o, Ceres, et LMA.

Troisième partie

MCSLAM : Un SLAM à contraintes multiples

Le SLAM estime simultanément la trajectoire d'un système de capteurs et le modèle de l'environnement (ou carte) qui l'entoure. Toutefois, ce processus échoue dans des conditions difficiles où les capteurs sont mis en défaut, ou subit une dérive liée à l'accumulation d'erreurs. Une voie d'amélioration consiste à utiliser des informations complémentaires sur le mouvement ainsi que le modèle d'environnement. Ces informations proviennent de capteurs supplémentaires ou d'*a priori* liés à l'application. Toutefois, l'utilisation d'une multitude d'informations pose des difficultés liées à la fusion de données et à l'estimation en temps réel de l'ensemble des paramètres du problème. La solution proposée dans cette partie prend la forme d'un *framework*, nommé MCSLAM, dédié à l'implémentation d'algorithmes de SLAM à contraintes multiples. Ce *framework* permet de fusionner des données capteurs à la fois hétérogènes et asynchrones, ainsi que des contraintes sur la structure 3D du modèle d'environnement reconstruit. Nous proposons une nouvelle implémentation des processus de localisation et de reconstruction permettant une estimation temps réel de la trajectoire. Notre solution s'appuie sur les travaux récents concernant les représentations continues de trajectoire ainsi que l'implémentation efficace de problèmes d'optimisation avec la bibliothèque LMA présentée dans la partie II.

Cette partie présente dans un premier temps la problématique liée à la réalisation d'application de SLAM contraint avec un état de l'art de ces méthodes. Puis l'implémentation du *framework* MCSLAM est présentée ainsi que les différentes contraintes de mouvement et de structure qui ont été réalisées. La partie se conclue avec des expérimentations mettant en évidence la modularité du *framework* ainsi que l'apport en terme de précision lié à l'utilisation de contraintes multiples dans le cadre d'applications temps réel de localisation.

Chapitre 8

SLAM contraint

Le SLAM contraint utilise les informations provenant de différents capteurs dans le processus d'estimation de la trajectoire (mouvement) et du modèle d'environnement (structure). L'exemple d'un tel système est illustré par la figure 8.1. Ainsi, on distingue deux catégories de contraintes : des contraintes sur le mouvement du système (modèle d'évolution, spline, odomètre, GPS, IMU, camera, ...) et des contraintes sur la structure de l'environnement (objets 3D connus ou partiellement connus, nuage de points 3D, carte de profondeur, ...). Naturellement, certaines contraintes comme l'erreur de reprojection ont une influence à la fois sur la trajectoire et la structure. De plus, les contraintes de structure impactent implicitement les paramètres de mouvement et inversement, car les contraintes lient les paramètres entre eux. Par exemple, l'environnement est observé par la caméra, donc un décalage du modèle d'environnement (par une contrainte de structure) provoque un décalage de la trajectoire dans le même sens. Ce chapitre présente dans un premier temps les approches les plus couramment utilisées dans l'état de l'art du SLAM contraint pour fusionner les données capteurs. Puis, la problématique ainsi que la solution apportée, sous la forme d'un *framework* seront abordées.

8.1 Etat de l'art du SLAM contraint

Les méthodes de SLAM monoculaire [53, 67, 3] reconstruisent à la fois la structure de l'environnement et la trajectoire de la caméra. Toutefois, ces méthodes subissent toujours une dérive du facteur d'échelle et une accumulation des erreurs [108]. Pour remédier à ces limitations, les méthodes de SLAM contraint (CSLAM) monoculaire utilisent des informations ou *a priori* supplémentaires (contraintes

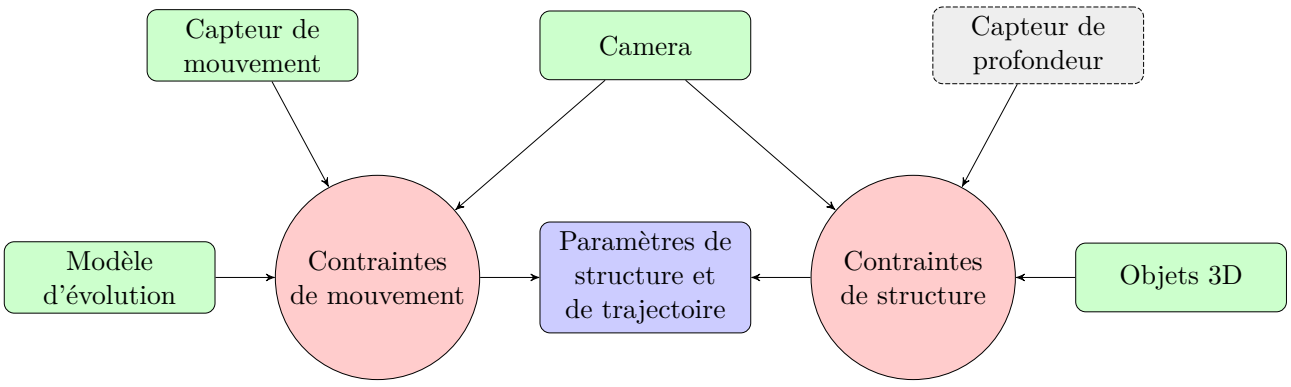


FIGURE 8.1 – L'algorithme MCSLAM est conçu pour être facilement configurable. Il peut gérer des configurations faisant intervenir différentes contraintes. Par exemple, un capteur de profondeur pourrait être intégré dans le système.

géométriques, modèles d'évolution, capteurs supplémentaires, ...) pour appliquer des contraintes sur la trajectoire de la caméra et la carte de l'environnement afin d'augmenter la précision de la reconstruction. De nombreux travaux ont été publiés sur le CSLAM durant la dernière décennie.

En ce qui concerne les contraintes sur la trajectoire, des travaux proposent d'utiliser des données de centrales inertielles et de GPS pour contraindre l'AF [109, 110, 111, 112, 113, 114, 115, 116, 117, 118]. Alors que Lhuillier [109] propose un nouvel algorithme de minimisation pour tenir compte de mesures hétérogènes, Kume *et al.* [110] et Micho *et al.* [111] minimisent une somme pondérée de données capteurs en plus de l'erreur de reprojection. Dans [116], une centrale inertielle est utilisée pour contraindre les poses d'un SLAM basé images clés. Cette approche calcule une interpolation linéaire des poses clés au temps où la donnée inertielle est mesurée pour appliquer la contrainte de mouvement. Dans [112], Kneip *et al.* intègrent les données d'un gyroscope afin d'apporter de la robustesse au processus de localisation. Un problème similaire est traité par Furgale *et al.* dans [117] en utilisant une représentation temporelle continue de la trajectoire afin de ne pas interpoler ou intégrer les données capteurs. En modélisant le mouvement des capteurs par une B-spline temporelle, ils ont démontré la possibilité d'estimer la transformation rigide entre la caméra et la centrale inertielle, ainsi que les biais de la centrale inertielle. Le même problème est résolu par Lovegrove *et al.* dans [118] en utilisant une B-spline cumulée pour définir la trajectoire d'un système de capteurs composé d'une caméra *rolling shutter* et d'une centrale inertielle. Tandis que Furgale *et al.* représentent la trajectoire par une spline de position et une spline d'orientation en utilisant la paramétrisation de Cayley-Gibbs-Rodriguez [119] (qui contient des singularités), Lovegrove *et al.* utilisent l'algèbre de Lie $\mathfrak{se}3$ du groupe des matrices $\mathbb{SE}3$ pour modéliser à la fois la position et l'orientation avec une seule spline.

Concernant les contraintes de structure, plusieurs travaux ont été menés. Pour des applications de Réalité Augmentée, Tamaazousti *et al.* [4] utilisent les observations provenant du modèle précis d'un objet observé. Ces observations sont ajoutées dans l'AF introduit par Mouragnon *et al.* [67] pour contraindre la position des points 3D du SLAM associés à l'objet. Toutefois, cette approche n'optimise pas la pose et la forme de l'objet au cours du SLAM. C'est pourquoi, même si le résultat de la méthode est très précis pour une contrainte impliquant un unique objet, celle-ci n'est pas utilisable pour des applications faisant intervenir plusieurs objets ayant été mal initialisés et modélisés. Le même principe a été appliqué par Larnaout et al. [114] dans le cadre de la localisation d'un véhicule en milieu urbain en utilisant des modèles approximatifs de bâtiments. En effet, les points 3D du SLAM représentant des façades de bâtiments sont contraints à appartenir à leurs façades respectives. Là aussi les modèles 3D utilisés ne sont pas optimisés, ce qui rend la précision de la localisation limitée à la précision des modèles utilisés. Pour des applications de reconnaissance et de suivi d'objets 3D, Galvez *et al.* [69] optimisent les paramètres d'objets 3D détectés en temps-réel dans l'environnement. La détection d'objet utilisée est basée sur un dictionnaire de sac de mots exhaustif utilisant le détecteur ORB. L'optimisation de l'ensemble des paramètres est réalisée avec la bibliothèque g2o. D'autres travaux utilisent des objets avec des systèmes stéréo multi-vues [120] ou avec des capteurs de profondeur [121, 122, 123, 124] pour reconstruire la forme 3D des objets. Dans [125], Dame *et al.* utilisent un SLAM dense pour construire une carte de profondeur à partir d'*a priori* 3D. Toutefois, cette solution nécessite des calculs supplémentaires comparativement aux approches éparses.

Le tableau 8.1 synthétise l'état de l'art présenté et positionne l'approche MCSLAM qui est décrite dans la suite de ce chapitre.

8.2 Problématique générale

Plusieurs éléments rendent difficiles l'implémentation d'algorithme de SLAM à contraintes multiples. Tout d'abord, les capteurs utilisés sont le plus souvent asynchrones et renvoient des informations de natures différentes : image, position, vitesse, accélération, données de profondeur... L'objectif étant d'estimer la trajectoire du système, une approche classique consiste alors à modéliser une trajectoire

	RT	Monoculaire	Mouvement	Structure	Continue
[120][123]				×	
[121]	×			×	
[124]			×	×	
[126][127]		×		×	
[128]	×	×		×	
[122]	×		×	×	
[125]		×	×	×	
[4][5]	×	×	×		
[69]	×	×	×	×	
[116]		×			
[117]					×
[118]		×			×
MCSLAM	×	×	×	×	×

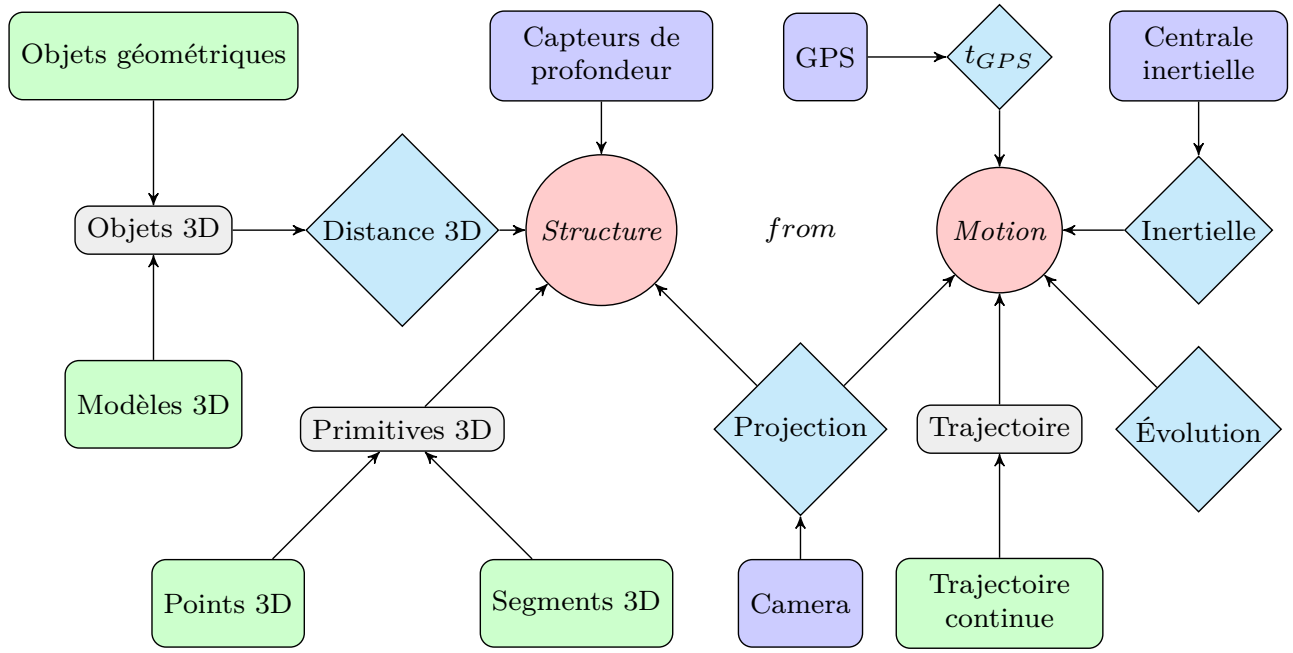
TABLE 8.1 – Résumé de l'état de l'art du SLAM contraint : le MCSLAM est la seule implémentation de SLAM monoculaire temps réel utilisant à la fois des contraintes de structure et de mouvement avec une trajectoire continue. RT = temps-réel, Mono = monoculaire, Mouvement : contraintes sur le mouvement, Structure = contraintes sur la structure, Continue = trajectoire continue.

discrète correspondant aux poses successives du système aux temps où les données capteurs sont acquises. Toutefois, le temps de traitement des données capteurs est potentiellement important (notamment pour les caméras). C'est pourquoi, on ne considère qu'un sous ensemble de ces données et donc un sous ensemble de poses afin d'assurer des performances temps réel, bien que cela dégrade la précision de localisation. La trajectoire est alors modélisée par un ensemble de poses de référence dont l'espacement temporel est un compromis entre le temps de traitement et la précision du résultat (comme par exemple l'utilisation des images clés dans le SLAM présenté dans le chapitre 3). De plus, les données sont potentiellement asynchrones et de natures différentes (pose, vitesse, accélération). Ainsi, l'autre inconvénient de cette modélisation est qu'elle ne permet pas de fusionner les données brutes des capteurs avec les poses de la trajectoire discrète à moins d'avoir recours à des approximations (*cf.* section 10.1).

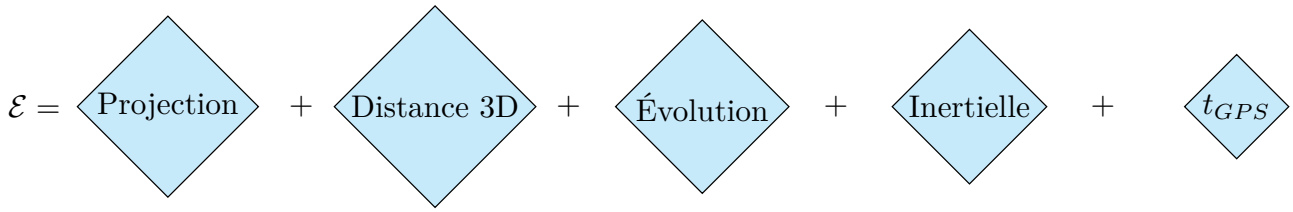
D'autres informations sont utilisées pour estimer la trajectoire. Il s'agit des *a priori* liés à l'application. Si le système de capteurs se déplace sur un plan, il est possible de supprimer un degré de liberté à la trajectoire ; ou si le système est fixé sur un véhicule, un modèle d'évolution améliorera l'estimation de la trajectoire. En supposant l'environnement rigide, des méthodes de reconstruction 3D par vision apporteront une meilleure localisation du système. De plus, il est possible que cet environnement contienne des objets connus ou partiellement connus. Cela permet une meilleure reconstruction 3D et donc une meilleure localisation. Ainsi, les connaissances *a priori* apportent une meilleure interprétation des données capteurs et apportent des contraintes supplémentaires au processus d'estimation de la trajectoire. Toutefois, le nombre de contraintes potentiellement important rend difficile l'implémentation du processus d'optimisation de la trajectoire.

La dernière problématique est la difficulté d'implémentation d'algorithmes performants, concernant le temps de calcul, de SLAM à contraintes multiples. En effet, les applications temps réel requièrent l'utilisation de code spécialisé (*cf.* 1.12.3) ce qui se fait en général au détriment de la modularité [1]. Ainsi les ajouts ou retraits de contraintes imposent des temps de développement relativement importants et conduisent à des codes très compliqués lorsque de nombreuses contraintes sont utilisées.

1. Capacité à s'adapter à différents types de problèmes.



(a) Exemple de problème d'optimisation de SLAM contraint.



(b) Fonction de coût du problème.

FIGURE 8.2 – Exemple de problème de SLAM à contraintes multiples.

Pour répondre à ces problématiques, un *framework* générique pour le développement d'applications performantes de SLAM à contraintes multiples a été développé. Il s'agit du MCSLAM qui est présenté dans la section suivante.

8.3 MCSLAM

Cette section présente la modélisation du MCSLAM. Il s'agit d'un *framework* basé sur la bibliothèque LMA (détaillée partie II) pour faciliter la création d'applications de SLAM contraint en fonction de configurations diverses (avec un nombre variable de contraintes et de paramètres). L'objectif de ce *framework* est d'offrir des performances d'exécution comparables à un code écrit à la main tout en étant modulaire, *i.e.* permettre d'ajouter ou de retirer facilement des contraintes au problème.

A titre d'exemple, la figure 8.2a illustre une application potentielle du MCSLAM. Plusieurs capteurs sont présents : une caméra, un capteur de mouvement et un capteur de profondeur. La caméra et le capteur de profondeur reconstruisent la structure de l'environnement en utilisant plusieurs types d'amers : des points 3D et des segments 3D (correspondant à des contours dans l'image). Pour améliorer la reconstruction de ces amers visuels, des objets 3D présents dans la scène sont conjointement estimés et les amers contraints à leurs surfaces. De plus, un capteur de profondeur amène une contrainte 3D supplémentaire sur les amers et les objets. L'estimation de la trajectoire est faite à la fois avec la caméra et avec des capteurs de mouvement (centrale inertielle et GPS). On peut également ajouter un

modèle d'évolution sur la trajectoire. La principale difficulté à l'implémentation de cette application est d'écrire un processus d'optimisation performant utilisant toutes ces contraintes.

On a vu que la bibliothèque LMA offre de bonnes performances. Mais cela se fait aux prix de la modularité car la fonction de coût doit être connue au moment de la compilation. Ainsi, chaque nouveau problème requiert une redéfinition explicite du solveur à la compilation ainsi que l'ajout des données dynamiques (insertion des paramètres à optimiser) à l'exécution. La solution proposée est d'ajouter un niveau d'abstraction, sans perte de performance, à la modélisation les processus d'optimisation (localisation et reconstruction) du SLAM contraint. Le MCSLAM est donc basé sur l'utilisation d'un graphe de dépendances (semblables à celui de la figure 8.2a), connu à la compilation, qui lie les paramètres aux contraintes. Ce graphe est analysé automatiquement pour déterminer la fonction de coût du problème correspondant à la somme des contraintes (illustrée par la figure 8.2b) pour générer le solveur adéquat avec LMA. A l'exécution du programme, le graphe est parcouru pour alimenter le solveur avec les paramètres à optimiser. La création d'une application de SLAM à contraintes multiples se fait alors par ajout de contraintes dans le graphe. Cette approche offre à la fois une vue d'ensemble sur la fonction de coût et permet de la modifier facilement (par ajout et suppression des contraintes dans le graphe). Pour chacune des contraintes, l'erreur minimisée correspond à la somme des observations sélectionnées pendant le parcours du graphe. La fonction d'erreur globale du problème correspond à la somme de toutes les contraintes :

$$\mathcal{E} = \sum_{i=1}^{\|\mathcal{C}\|} \sum_{k=1}^{\|\mathcal{C}_i\|} \left\| \frac{\mathcal{C}_{i,k}}{\sigma_i} \right\|^2 \tag{8.1}$$

avec $\mathcal{C}$ les différents types de contraintes, avec K_i le nombre d'observations correspondant à la contrainte i, $\mathcal{C}_{i,k}$ l'erreur associée à l'observation k de la contrainte i et σ_i l'estimation de l'erreur de mesure (qui est spécifique à chaque contrainte). L'avantage de cette approche est que la fonction de coût $\mathcal{E}$ est re-générée automatiquement en analysant le graphe si celui-ci change.

La conception du *framework* MCSLAM est basée sur 3 contributions :

1. la bibliothèque d'optimisation LMA présentée dans le chapitre II,
2. une architecture logicielle avec un haut niveau de modularité et de performances,
3. une utilisation temps réel d'un modèle de trajectoire paramétrique offrant une manière efficace d'inclure des contraintes hétérogènes asynchrones dans le processus d'optimisation (détaillée chapitre 10).

Les objectifs de l'approche MCSLAM et les solutions proposées sont :

— la modularité : utilisation d'un graphe de dépendances entre paramètres et contraintes ; ajouter ou retirer une contrainte revient alors à l'ajout ou la suppression d'un nœud dans le graphe,
— la performance : résolution du problème d'optimisation basée sur l'utilisation de LMA.

8.3.1 Vue générale

Le MCSLAM est doté d'une architecture classique de SLAM. Ainsi, trois processus principaux sont exécutés en parallèle : l'acquisition, la localisation et la reconstruction. La « reconstruction » est une notion relative au SLAM visuel. Dans un contexte plus général, on parle ici de reconstruction pour qualifier l'optimisation de l'ensemble des paramètres du problème par opposition à la localisation où seule la trajectoire est optimisée. Le processus d'acquisition est composé d'une liste de capteurs gérés par un outil d'acquisition multi-capteurs nommé vbus et disponible dans la bibliothèque libv [2]. La localisation reçoit et traite les données pour estimer la pose courante en fonction de différentes contraintes. Toutefois, seuls les paramètres de trajectoire sont optimisés (la carte 3D n'est pas mise à

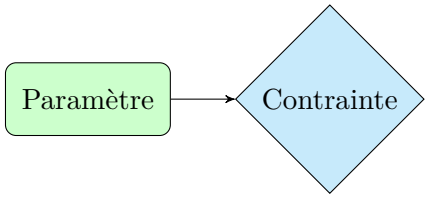
2. http://git.univ-bpclermont.fr/libv

jour). Lorsque cela est nécessaire (*e.g.* à l'ajout d'une image clé), le processus de reconstruction est appelé. Celui-ci utilise les données récemment acquises pour mettre à jour la carte et effectuer une optimisation des paramètres du problème en fonction de multiples contraintes.

L'originalité du MCSLAM réside dans la modélisation des processus d'optimisation (localisation et reconstruction) sous forme de graphe afin de faciliter le développement. En effet, ajouter une contrainte revient alors à ajouter un élément dans le graphe. Le MCSLAM prend en charge la mise à jour du solveur afin de tenir compte des nouvelles données (modification de la fonction de coût, nouveaux paramètres à optimiser ...).

8.3.2 Graphe de dépendances

La modularité du MCSLAM provient de la modélisation des problèmes d'optimisation sous la forme d'un graphe de dépendances. Ce graphe est composé de deux types de nœuds : des nœuds de contraintes liés à des nœuds de paramètres. Dans la suite du manuscrit, la représentation graphique d'un nœud qui dépend d'une contrainte est la suivante :



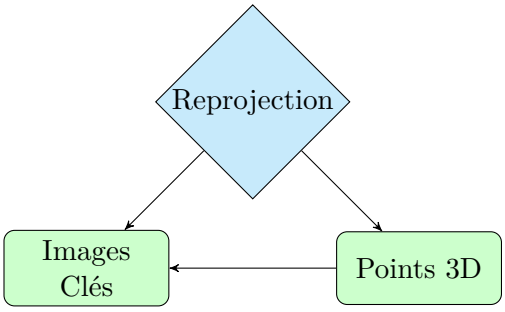
Les liens sont orientés afin de définir les dépendances entre les nœuds du graphe. Chaque nœud est défini par l'utilisateur et doit contenir les éléments suivants :

- pour les nœuds de paramètres :
  - à la compilation : le type de paramètre du nœud,
  - à l'exécution :
    - les dépendances (autres nœuds du graphe),
    - les instances de paramètres à optimiser,
- pour les nœuds de contraintes :
  - à la compilation : la fonction de coût du nœud,
  - à l'exécution :
    - les dépendances,
    - les instances d'erreurs du nœud à ajouter à l'optimisation.

Le graphe est parcouru de deux façons :

1. à la compilation : pour lister les différents termes d'erreurs de la fonction de coût du problème afin de générer un solveur spécifique avec la bibliothèque LMA,
2. à l'exécution : pour parcourir le graphe afin de définir les paramètres à optimiser et les contraintes nécessaires tout en respectant l'ordre de parcours défini par les dépendances.

A titre d'exemple, pour un problème de SLAM classique, la définition du graphe de dépendances suivant :



nécessite l'implémentation par l'utilisateur de 3 nœuds ayant les caractéristiques suivantes :

- nœud de paramètres « Images clés » :
  - le type de paramètre à optimiser est une pose à 6 *dof.* (paramétrisation classique),
  - les poses à optimiser sont les 3 dernières images clés (critère de Mouragnon vu dans la section 2.2).
- nœud de paramètres « Points 3D » :
  - le type de paramètre à optimiser est un point 3D à 3 *dof.*,
  - le nœud « Images clés » doit avoir été préalablement appelé,
  - les points 3D à optimiser sont ceux vus dans les images clés à optimiser (définies par le nœud « Images clés »).
- nœud de contraintes « Reprojection » :
  - la contrainte est l'erreur de reprojection entre des poses et des points 3D,
  - les nœuds « Images clés » et « Points 3D » doivent avoir été préalablement appelés,
  - la liste des erreurs est définie par l'ensemble des points 3D définis dans le noeud « Points 3D » projeté dans l'ensemble des poses 3D définies dans le noeud « Images Clés ».

A la compilation, un solveur sera généré pour minimiser la contrainte d'erreur de reprojection, et à l'exécution, le graphe sera parcouru dans l'ordre « Images clés » → « Points 3D » → « Reprojection » On observe que le nœud « Reprojection » dépend de la résolution préalable des nœuds de paramètres. C'est pourquoi, la notion de dépendance du graphe est importante. Dans la suite du manuscrit, les schémas représentant les graphes de dépendances pourront ne pas contenir l'ensemble des liens entre nœuds afin de faciliter la lecture.

La forme générale du graphe doit être définie statiquement, *i.e.* directement dans le code source et prend une forme relativement simple :

```
Graph<Constraints(Reprojection),Parameters(KeyFrames,Points3D)> graph;
```

Les dépendances entre nœud n'apparaissent pas dans la définition statique du graphe car elles ne sont utiles qu'à l'exécution du programme. L'ajout ou le retrait d'un nœud (contraintes ou paramètres) s'effectue simplement en ajoutant ou retirant le nœud dans les listes **Constraints** ou **Parameters**. La sélection des paramètres à optimiser et les erreurs à utiliser sont définies à l'exécution. Ainsi conçu, le MCSLAM permet de générer facilement des configurations diverses de SLAM. Le graphe de dépendances ajoute un niveau d'abstraction[3] à l'utilisation de la bibliothèque LMA. C'est pourquoi, celui-ci est disponible en *open-source* avec la bibliothèque LMA.

8.4 Conclusion

L'état de l'art du SLAM contraint est riche en contributions. Toutefois, chaque travail de recherche se focalise sur le développement d'une ou deux contraintes. L'objectif du MCSLAM est de proposer une approche suffisamment modulaire et performante pour intégrer au sein d'un même algorithme les travaux de l'état de l'art. La configuration du graphe de dépendances se fait simplement par ajout ou retrait des contraintes, ce qui permet de générer rapidement différentes configurations de SLAM. Dans le cadre de la thèse, le *framework* a été utilisé pour générer des applications de SLAM visuel, telles que l'algorithme présenté dans le chapitre 3 et des applications à contraintes multiples, comme le montre les expérimentations présentées dans le chapitre 11.

3. Cette abstraction est résolue à la compilation donc elle ne pénalise pas les performances d'exécution.

Chapitre 9

Contraintes de structure

Une des problématiques abordées au cours de ce travail traite du recalage d'objets pour des applications de Réalité Augmentée. L'idée est d'utiliser une caméra vidéo pour observer l'environnement et, en temps réel, superposer une information virtuelle sur un objet physique. On parle alors de « recalage » d'objet. Le résultat de la superposition est visualisé à travers un support : écran d'ordinateur, tablette tactile, lunettes de Réalité Augmentée... Pour y parvenir, il est nécessaire d'estimer la pose de l'objet par rapport à la caméra. De plus, le recalage doit fonctionner même lorsque l'objet est peu ou pas observé. Cela nécessite un référencement 3D de l'objet par rapport à la caméra ainsi qu'une localisation de la caméra par rapport à l'environnement. Les méthodes de SLAM dans ce contexte sont intéressantes car elles estiment la pose de la caméra par rapport à l'environnement. De plus, la carte d'amers 3D reconstruite en temps réel est utilisée pour estimer la pose de l'objet. Et l'objet apporte une information sur la position des amers 3D. Ainsi, chaque objet recalé constitue une contrainte sur le modèle d'environnement du SLAM. Ces contraintes sont également utilisées pour estimer précisément la pose et la forme des objets. Il existe plusieurs types de contraintes : des contraintes d'apparences des contraintes point à point, point à surface, voir même des contraintes sur la profondeur des points. L'utilisation de contraintes sur l'apparence des objets n'a pas été traité dans ce travail.

9.1 Problématique

La problématique abordée concerne l'efficacité du recalage, à la fois sur la précision que sur le temps de calcul, entre les modèles virtuels et les objets réels. On suppose que dans un contexte de Réalité Augmentée, les poses et les dimensions des objets ne sont pas connues à l'avance. De plus, la solution proposée doit fonctionner simultanément avec plusieurs objets.

Ce chapitre liste les différentes contraintes qui ont été implémentées pour améliorer la reconstruction de la carte 3D du SLAM. Ces contraintes se présentent sous forme d'objets 3D présents dans l'environnement. Dans ce travail, on dispose d'une liste prédéfinie d'objets de formes simples et complexes. Les contraintes liées aux objets sont initialisées en ligne au fur et à mesure que les objets sont détectés (la détection pouvant être automatique ou manuelle). De plus, la pose et la dimension de chacun des objets sont optimisées avec les autres paramètres du SLAM. Cela est nécessaire pour pouvoir traiter plusieurs objets 3D grossièrement initialisés en ligne. Lorsque qu'un seul objet est utilisé comme contrainte, il est possible de ne pas optimiser sa pose (c'est l'approche utilisée dans [129, 130]). En effet, l'optimisation aura tendance à corriger la trajectoire de la caméra pour expliquer au mieux l'hypothèse de l'objet. Toutefois, cela nécessite plusieurs optimisations car d'autres contraintes sont appliquées à la trajectoire (par exemple l'ancrage des poses du SLAM via l'erreur de reprojection). Les contraintes présentées sont simples. Elle consiste à minimiser des distances 3D entre les amers reconstruits par le SLAM et les objets 3D. Des contraintes plus élaborées peuvent être utilisées. Par exemple, en supposant que les points ne puissent se déplacer qu'à la surface des objets afin de sup-

primer un degré de liberté par point, comme le proposent Tamaazousti *et al.* [130]. Il est également possible d'utiliser l'apparence photométrique des objets. Toutefois, on souhaite montrer que l'utilisation de contraintes basiques au sein du MCSLAM est suffisante pour donner une bonne estimation de la pose des objets et donc de la reconstruction. D'autres contraintes sur la structure pourraient provenir d'un capteur de profondeur mais aucune expérimentation dans ce sens n'a été réalisée.

Pour résumer, l'objectif est de proposer une solution précise et performante pour des applications de Réalité Augmentée avec recalage simultané d'objets multiples, statiques et rigides étant initialisés en ligne et approximativement connus.

9.2 Gestion des objets 3D

Cette section décrit la gestion des objets 3D utilisés comme contraintes dans le processus d'optimisation du SLAM. La gestion des objets est dynamique, afin que la présence d'un objet ne constitue pas une condition indispensable à la localisation de la caméra. Ainsi les objets sont initialisés, utilisés comme contrainte et supprimés en ligne (ce qui est présenté dans la section suivante 9.2.1). Afin d'éviter toute confusion, il faut noter que cette gestion dynamique ne va pas créer ou supprimer des nœuds dans le graphe de dépendances présenté section 8.3.2. Les nœuds correspondant aux objets 3D et les types de contraintes associés sont définis à la compilation. Toutefois, le nombre d'instances d'objets 3D dans les nœuds ainsi que le nombre de contraintes associées (associations entre les amers 3D et les objets) évoluent dynamiquement. Les associations entre objets et amers 3D sont effectuées selon un critère de distance 3D présenté dans la section 9.2.2. Le critère 3D apporte plus de robustesse par rapport à un critère d'association effectué dans l'image dans le cas où les objets sont occultés (ils peuvent même s'occulter entre eux *cf.* 11.1) ou si l'arrière plan de l'objet est très texturé. Toutefois, rien n'empêche de compléter le critère 3D par un critère 2D qui testerait l'appartenance du point d'intérêt à la projection du modèle dans l'image.

9.2.1 Initialisation et suppression des objets.

L'initialisation consiste à recaler, approximativement, le modèle 3D d'un objet sur un sous-ensemble de points provenant de la reconstruction 3D. Pour cela, il est nécessaire de s'abstraire de 3 problèmes difficiles :

1. y'a-t-il un objet connu dans l'image ou dans le nuage de points 3D reconstruit ?
2. quel est le type de cet objet ?
3. quels points d'intérêt dans l'image, ou quels points 3D, font parti de l'objet ?

Ces problèmes n'ont pas été abordés dans ce travail [1]. C'est pourquoi, une procédure semi-automatique est utilisée. Durant le SLAM, la position de l'objet (à initialiser) dans l'image est définie par l'utilisateur. Celui-ci utilise l'interface graphique pour sélectionner l'enveloppe convexe de l'objet. Puis, les points d'intérêt à l'intérieur de l'enveloppe convexe sont utilisés pour initialiser l'objet à partir des associations 2D−3D. Pour chaque type d'objet, une heuristique différente est utilisée pour déterminer les paramètres. Si aucun point 3D de l'environnement n'est associé à un objet 3D, celui-ci n'apporte plus de contrainte. Il est donc possible de le supprimer du processus d'optimisation. L'association entre un amer 3D et un objet est décrite à la section suivante.

1. Des algorithmes classiques de détection d'objets dans des images 2D ou des méthodes à base de sac de mots *cf.* [69] pourraient être utilisés pour obtenir une initialisation pleinement automatique.

9.2.2 Association entre objets et amers 3D

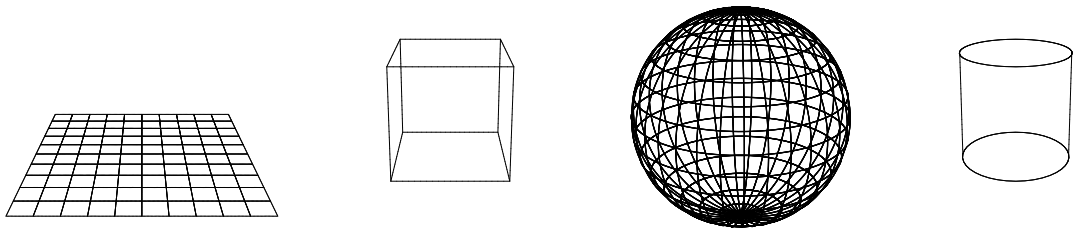
Les contraintes provenant des objets 3D sont intégrées dynamiquement dans le processus de mise à jour de la carte du SLAM[2]. Toutefois, les points 3D reconstruits peuvent correspondre soit à la partie inconnue de l'environnement, soit aux objets partiellement connus. Pour assurer la convergence de l'optimisation, une étape d'association entre les points 3D et les objets correspondants est réalisée avant chaque optimisation. Dans ce travail, un point 3D P est associé à un objet Π si la distance $d_\perp$ entre P et la plus proche surface de l'objet Π est inférieure à un seuil σ_Π. Pour choisir une valeur de σ_Π invariante au facteur d'échelle de la reconstruction, la dimension D_Π de chaque objet Π est utilisée telle que $\sigma_\Pi = D_\Pi \times \lambda$ avec λ l'intensité de la contrainte (on utilise 0.02). Si une association est créée, les paramètres des points 3D P et les paramètres de l'objet correspondant Π sont optimisés afin de minimiser la distance $d_\perp$. Ainsi, le processus d'optimisation doit combiner des erreurs pixeliques (erreurs de reprojection) et des erreurs métriques (distances 3D entre les points et les objets). L'homogénéisation de ces erreurs est possible en pondérant les erreurs 3D par le coefficient σ_Π. L'association entre un objet et un amer 3D est donc réalisée sur un critère de distance 3D. Toutefois, lorsque cela est rapide et trivial à calculer (par exemple pour le plan et le parallélépipède rectangle définis section suivante), un critère d'appartenance à l'enveloppe convexe de l'objet dans l'image est utilisé en complément du critère de distance 3D. L'utilisation d'un critère d'association 3D peut poser problème lorsque les objets sont proches les uns des autres. Une telle configuration peut nécessiter, en plus de l'utilisation du critère d'appartenance à l'enveloppe convexe, l'utilisation d'un M-Estimateur pour marginaliser implicitement les mauvaises associations dans l'optimisation.

Il est possible qu'au cours de la séquence, l'objet soit toujours dans le champ de la camera mais de moins en moins observé (à cause d'un éloignement de la caméra par exemple). Admettons par exemple qu'un objet ne soit associé qu'à un seul point 3D. Il suffirait que ce point 3D ne soit pas parfait pour que l'objet soit décalé de sa position par l'optimisation (pour coller au mauvais point 3D). Ainsi le recalage serait erroné. C'est pourquoi, utiliser uniquement les points 3D observés à l'instant courant peut poser problème. La solution qui s'est révélée efficace en pratique consiste à compléter les associations en utilisant un historique de points 3D qui ont été préalablement associés. Ainsi, chaque objet possède son propre historique de points 3D dont la dimension est limitée aux 250 derniers points 3D (les 250 plus récentes associations). De plus, lorsque les points 3D provenant de l'historique n'appartiennent plus à la fenêtre d'optimisation incrémentale des paramètres définie section 3.5, ceux-ci sont considérés avec 0 *ddl.* ce qui permet de fixer l'objet 3D et contribue à limiter la dérive de la reconstruction.

9.3 Objets 3D simples

La figure 9.1 illustre les objets de formes basiques qui ont été modélisés dans ce travail. Il s'agit d'un plan, d'une sphère, d'un parallélépipède rectangle et d'un cylindre. Chaque objet est initialisé spécifiquement à partir de la liste d'associations de points 2D-3D sélectionnés manuellement (étape décrite section 9.2.1). Les contraintes associées à ces objets minimisent la distance 3D entre le point dans le repère monde et la surface de l'objet associé. Le calcul de distance entre un point 3D et ces objets est trivial. Mis à part pour le plan, il suffit d'exprimer le point 3D dans le repère de l'objet, puis le calcul de distance est direct. Toutefois, une étape d'association à la bonne face peut être nécessaire (pour le cube et le cylindre). Les sections qui suivent présentent les particularités liées à chaque objet (*ddl.*, calcul de distance, ...). Afin de simplifier l'écriture, chaque objet sera noté Π et la pose R, t. La valeur σ_Π est définie précédemment 9.2.2.

2. les types des contraintes sont définis à la compilation dans la processus d'optimisation, mais le nombre de contraintes d'un même type dépend des données dynamiques du SLAM.



(a) Plan (b) Parallélépipède (c) Sphère (d) Cylindre

FIGURE 9.1 – Objets simples utilisés comme contraintes sur la structure 3D.

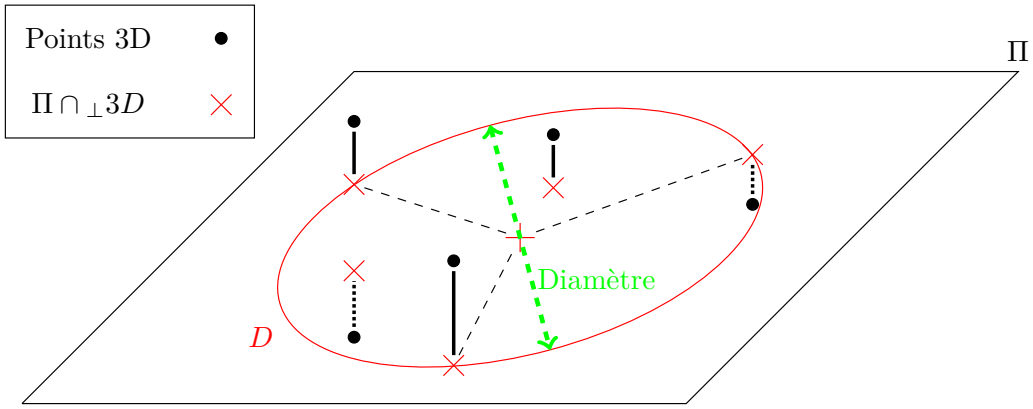


FIGURE 9.2 – Contrainte planaire

9.3.1 Plan

Les objets de type « Plan » appliquent une contrainte planaire sur les amers 3D. Un plan est paramétré par sa normale, donc 3 *ddl.* sont nécessaires (le choix de la paramétrisation est détaillé dans l'annexe 11.7). Pour initialiser un plan, on utilise l'ensemble des points 3D présents dans l'enveloppe convexe définie manuellement, pour estimer la normale avec une décomposition en valeurs singulières [131]. La distance entre un point 3D dans le repère monde P et un plan noté Π est obtenue en utilisant les 4 coefficients de l'équation du plan $\Pi = [a, b, c, d]$ telle que :

$$d_{\perp}(P, \Pi) = [P, 1] \times \Pi^T \tag{9.1}$$

Ainsi, la contrainte planaire minimise la somme des distances entre l'ensemble des points 3D de $\mathcal{P}$ et le plan ce qui est formalisé par :

$$\mathcal{E}(\mathcal{P}, Plan) = \sum_{i=0}^{\|\mathcal{P}\|} \left(\frac{d_{\perp}(\mathcal{P}_i, \Pi)}{\sigma_{\Pi}} \right) \tag{9.2}$$

Afin de pondérer l'erreur 3D, on définit la dimension du plan comme étant le diamètre du disque englobant les projections orthogonales des points 3D sur le plan, comme illustré sur la figure 9.2.

9.3.2 Sphère

Les objets de type « Sphère » appliquent une contrainte sphérique sur les points 3D. Une sphère est paramétrée par une position t et un rayon : $\Pi = \{t, rayon\}$ donc 4 *ddl.* sont nécessaires. Pour initialiser une sphère, on utilise la moyenne des points 3D de $\mathcal{P}$ pour définir la position, et l'écart

type de $\mathcal{P}$ pour le rayon. Le critère minimisé par la contrainte sphérique correspond à la somme des distances entre les points 3D de $\mathcal{P}$ et la surface de la sphère ainsi définie :

$$\mathcal{E}(\mathcal{P}, \Pi) = \sum_{i=0}^{\|\mathcal{P}\|} \left(\frac{\|t - \mathcal{P}_i\| - rayon}{\sigma_\Pi} \right) \tag{9.3}$$

La dimension de la sphère, utilisée pour définir σ_Π, est définie par le diamètre.

9.3.3 Parallélépipède rectangle

Les objets de type « parallélépipède rectangle » appliquent une contrainte planaire qui est fonction de l'association aux faces. Le parallélépipède rectangle est paramétré par 9 *ddl.* : $\Pi = \{R, t, l, p, h\}$ avec R l'orientation, t la position du centre (intersection des diagonales entre sommets opposés) et l, p, h les paramètres de longueur, profondeur et hauteur. La position t est initialisée à la moyenne des points de $\mathcal{P}$ et les dimensions sont estimées à partir des coordonnées minimales et maximales des points de $\mathcal{P}$ telles que :

$$\begin{cases} inf = \{\min_x(\mathcal{P}), \min_y(\mathcal{P}), \min_z(\mathcal{P})\} \\ sup = \{\max_x(\mathcal{P}), \max_y(\mathcal{P}), \max_z(\mathcal{P})\} \\ \{l, p, h\} = inf/2 - sup/2 \end{cases}$$

L'étape d'association d'un point 3D à la bonne face du parallélépipède est réalisée de la manière suivante :

1. recherche du sommet noté **s** étant le plus proche des 8 sommets du parallélépipède,
2. sélection des 3 faces du parallélépipède qui ont le sommet **s** en commun,
3. le point P, dont la dernière observation par la caméra est le point d'intérêt p, est associé à la face **f** si p est dans l'enveloppe convexe de **f**.

Une fois la face sélectionnée, la distance $d_\perp(\Pi, P_c)$ entre le point 3D P_Π exprimée dans le repère de l'objet correspond plus ou moins à une des composantes de P_Π. Le critère minimisé ici correspond à la somme des distances entre les points 3D de $\mathcal{P}$ et les faces (préalablement sélectionnées) du parallélépipède :

$$\mathcal{E}(\mathcal{P}, \Pi) = \sum_{i=0}^{\|\mathcal{P}\|} \left(\frac{d_\Pi(R^T(\mathcal{P}_i - t))}{\sigma_\Pi} \right) \tag{9.4}$$

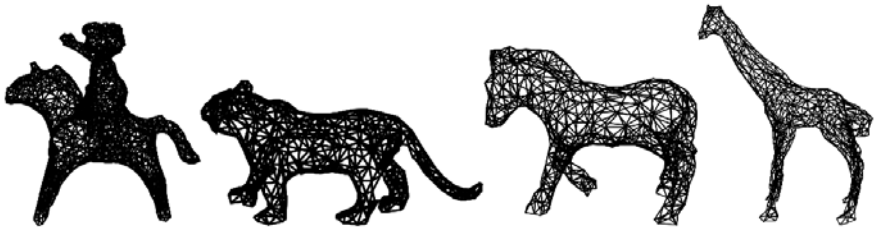
9.3.4 Cylindre

Les objets de type « Cylindre » appliquent des contraintes planaires pour les points associés aux bases (disques situés aux extrémités du cylindre) et cylindrique pour les points du contour. Un cylindre est paramétré avec 7 *ddl.* : $\Pi = \{R, t, rayon, hauteur\}$ avec R l'orientation, t la position de la base supérieure. Le rayon est pris sur la base, et la hauteur est la distance entre les deux bases. On ne considère que 2 *ddl.* pour la rotation. La distance entre un point P_Π, exprimé dans le repère du cylindre, et la surface du cylindre est obtenue en calculant la distance minimale aux bases et au contour :

$$d_\perp(P, \Pi) = \min(|\|P\| - rayon|, |P_y|, |P_y - hauteur|); \tag{9.5}$$

Le critère minimisé correspond à la somme des distances entre les points 3D de $\mathcal{P}$ et le cylindre :

$$\mathcal{E}(\mathcal{P}, \Pi) = \sum_{i=0}^{\|\mathcal{P}\|} \left(\frac{d_\perp(R^T(\mathcal{P}_i - t), \Pi)}{\sigma_\Pi} \right) \tag{9.6}$$



(a) Résultats de la photo-modélisation d'objets « complexes », avec le logiciel CMPSMVS, utilisés comme contraintes sur la structure 3D.

(b) Objets « complexes » utilisés pour la photo-modélisation.

FIGURE 9.3 – Objets utilisés pour des applications de Réalité Augmentée.

9.4 Modèle 3D photo modélisé

Afin de gérer des objets de formes plus complexes, on a eu recours à la photo-modélisation. Pour cela, on effectue préalablement une série de photos de l'objet à modéliser, puis le logiciel CMPMVS est utilisé pour générer un modèle d'objet. La figure 9.3 présente quelques résultats de photo-modélisation utilisés dans les expérimentations. Seuls les sommets des modèles générés sont utilisés (l'information de texture n'est pas considérée). De plus, le modèle est reconstruit à un facteur d'échelle près. C'est pourquoi, on considère le facteur d'échelle de l'objet comme un *ddl.* à estimer. On compte alors 7 *ddl.*, 6 pour la pose et 1 pour l'échelle. Pour calculer la distance d'un point 3D à l'objet, on exprime le point P_w dans le repère de l'objet :

$$\mathcal{E}(\mathcal{P}, \Pi) = \sum_{i=0}^{\|\mathcal{P}\|} \left(\frac{d_\perp(R^T(\mathcal{P}_i - t)/s, \Pi)}{\sigma_\Pi} \right) \tag{9.7}$$

avec R et t l'orientation et la position de l'objet, s correspond au facteur d'échelle. Puis, la fonction $d_{(.)}$ consiste à chercher le point de la surface de l'objet qui est le plus proche du point P_c ($P_w$dans le repère objet). Le modèle étant potentiellement composé d'un grand nombre de sommets (plusieurs milliers dans les expérimentations), on utilise le *kd-tree* de la bibliothèque FLANN [132] : un arbre de recherche est généré une unique fois à partir des sommets de l'objet dans son repère local. Le fait d'exprimer le point 3D P_w dans le repère de l'objet permet de ne pas générer à nouveau l'arbre de recherche avec FLANN à chaque fois que les paramètres de l'objet sont mis à jour pendant le processus de reconstruction. Ayant à disposition des maillages relativement denses, on utilise l'arbre de recherche pour obtenir les 3 points du modèle les plus proches de P_c. Le critère à minimiser correspond à la distance $d_\perp$ entre le barycentre de ces 3 points et P_c. Cette approche est un compromis entre temps de calcul et précision. D'autres stratégies sont envisageables. Une approche judicieuse serait d'ajuster dynamiquement la densité du maillage en fonction de la distance à l'objet comme le fait Chevaldonné *et al.* [133].

9.5 Conclusion

Les contraintes présentées dans ce chapitre permettent à la fois d'estimer la pose des objets 3D et de contraindre le processus de cartographie. Des expérimentations appliquées à la Réalité Augmentée dans la section 11.1 mettent en évidence l'efficacité de l'approche avec des contraintes de structure relativement simples. Une problématique intéressante serait d'évaluer l'approche avec des objets de grandes dimensions, comme un bâtiment ou une ville. Dans le cadre d'applications de visites touristiques guidées, il serait même envisageable d'évoluer à l'intérieur de l'objet utilisé comme contrainte. Par exemple, en se déplaçant dans un château préalablement photo-modélisé afin d'observer en Réalité Augmentée les pièces dans leurs états originaux.

Certaines problématiques n'ont pas été abordées, comme par exemple l'initialisation automatique des objets. La difficulté est de détecter correctement le type et la position de l'objet mais également de gérer les mauvaises initialisations. En effet, une erreur d'initialisation d'objets 3D (initialisation du mauvais type d'objet par exemple) appliquera une mauvaise contrainte qui dégradera la reconstruction 3D et dont la conséquence peut être l'échec de la localisation. Une solution envisageable serait d'ajuster la pondération σ_Π associée à l'objet Π en fonction de la confiance attribuée à la détection et de la faire évoluer en fonction de l'erreur $\mathcal{E}(\mathcal{P}, \Pi)$.

Chapitre 10

Contraintes de mouvement

Ce chapitre présente les contraintes de mouvement utilisées dans le MCSLAM pour améliorer le processus d'estimation de la trajectoire. Ces contraintes proviennent de plusieurs sources : capteurs, modèle d'évolution, contraintes indirectes liées à l'observation de l'environnement ... Elles sont donc de natures différentes et asynchrones. C'est pourquoi, on a recours à une modélisation particulière de la trajectoire afin de fusionner ces données tout en conservant des performances temps réel. Dans un premier temps, la problématique liée à la fusion de données hétérogènes et asynchrones est présentée. Ensuite, le modèle de trajectoire que l'on a choisi d'utiliser est détaillé et adapté pour être utilisé dans le cadre d'applications temps réel. Puis des contraintes de mouvement provenant de différents capteurs ainsi qu'un modèle d'évolution simple sont présentés.

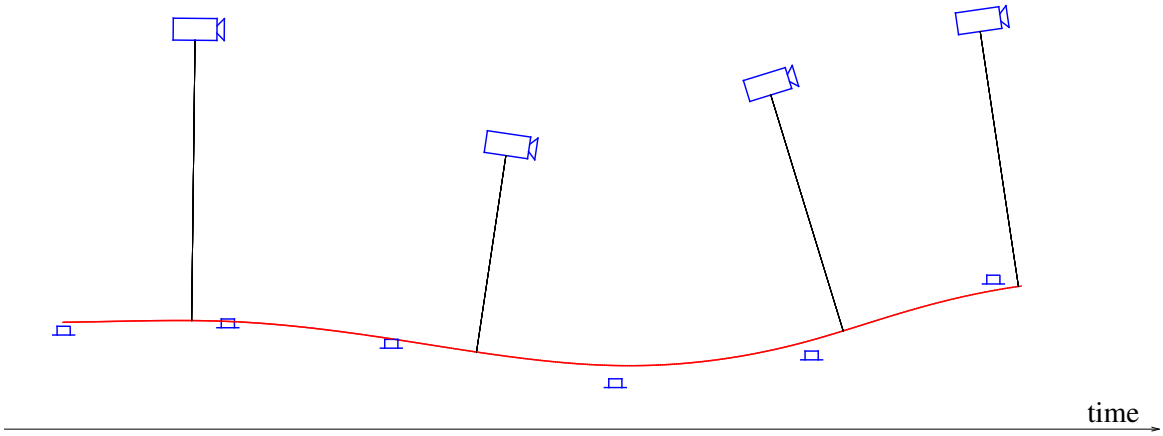
10.1 Problématique

L'utilisation de plusieurs capteurs améliore les méthodes de SLAM. Ceux-ci apportent des informations redondantes voir complémentaires. Par exemple, la localisation par caméra est connue pour dériver malgré un excellent positionnement relatif entre deux images successives. Même en utilisant plusieurs caméras à champs recouvrant, la dérive ne peut pas être totalement corrigée sur de grands parcours. Tandis qu'un GPS bas coûts donnera un positionnement approximatif mais ne subira aucune dérive quelle que soit la distance parcourue. C'est pourquoi, l'utilisation de plusieurs capteurs complémentaires est une solution intéressante dans le cadre d'applications de SLAM.

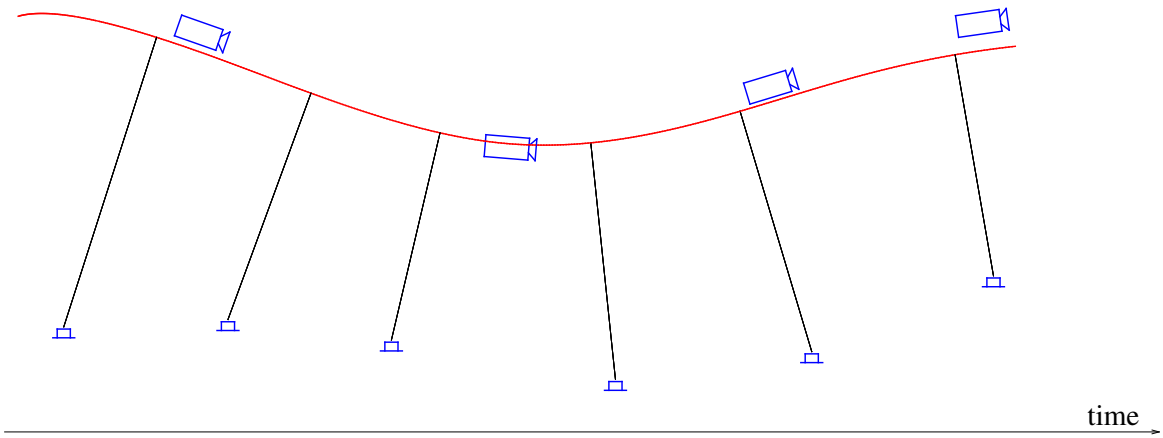
Afin de réaliser de telles applications, il est nécessaire de définir un modèle de trajectoire adapté à la fusion de données des différents capteurs (quelques méthodes de l'état de l'art pour la fusion par optimisation sont brièvement rappelées dans la section 8.1).

Par exemple, considérons le problème d'estimation de la trajectoire d'un système de capteurs asynchrones composé d'une caméra et d'une centrale inertielle (rigidement liées). Une solution, illustrée par la figure 10.1a, consiste à utiliser une trajectoire discrète composée des poses successives de la caméra. Pour utiliser les données inertielles afin de contraindre les poses caméra de la trajectoire, il est nécessaire d'interpoler les données inertielles aux temps des poses caméra. De plus, les données inertielles correspondant à des vitesses de rotation et des accélérations de position, celles-ci sont intégrées respectivement 1 et 2 fois pour contraindre les poses caméra. Or, l'erreur liée au bruit des capteurs augmente de manière quadratique lorsque les données sont intégrées. Sachant que les données inertielles brutes sont imprécises, effectuer des intégrations, puis des interpolations dégrade fortement la précision du résultat.

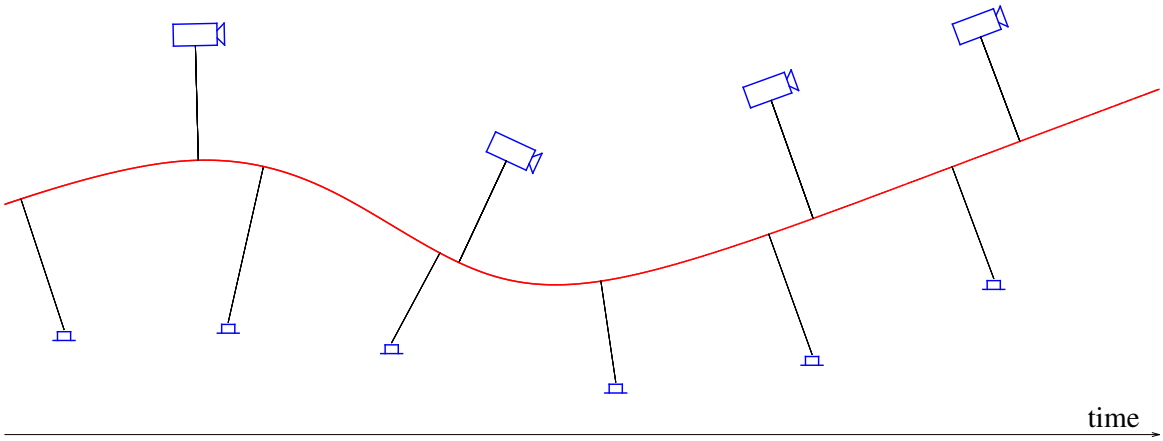
Une autre approche, illustrée par la figure 10.1b, consiste à interpoler la pose des caméras, au temps des données inertielles. Dans cette approche, la qualité de l'interpolation des poses caméra dépend de la répartition des poses (uniformément réparties et rapprochées). Or dans les applications de SLAM temps-réel, il est nécessaire de sous-échantillonner les poses et donc d'utiliser des poses



(a) Interpolation et intégration des données inertielles.



(b) Interpolation et dérivation des poses caméra.



(c) Contraintes inertielles et caméras appliquées sur une trajectoire continue.

FIGURE 10.1 – Différentes approches pour fusionner des données inertielles avec une caméra.

clés. Ainsi, l'interpolation est d'autant plus imprécise que ces poses clés sont éloignées. De plus, le fonctionnement d'un tel système dépend principalement de la localisation par vision. Si pour une raison quelconque, celle-ci échoue, alors le processus d'optimisation de la trajectoire basé images clés ne pourra pas s'effectuer. C'est pourquoi dans une telle approche, il est impossible d'effectuer la fusion de données lorsque la localisation échoue.

Une solution robuste nécessite que tous les capteurs collaborent à l'estimation d'une même trajectoire sans qu'aucun d'entre eux ne soit indispensable. De plus, il faut éviter les interpolations et intégrations qui dégradent la qualité des données capteurs. Une solution issue de travaux récents [117, 118] et illustrée par la figure 10.1c, consiste à utiliser une représentation continue et fonction du temps de la trajectoire. Celle-ci est alors contrainte par l'ensemble des informations capteurs. Cette représentation est 2 fois dérivable ce qui rend possible l'utilisation de données inertielles. En effet, il est possible d'évaluer la vitesse et l'accélération en tout temps de la trajectoire en dérivant celle-ci. Ce modèle de trajectoire est décrit dans la section suivante.

10.2 Trajectoire continue

Afin d'estimer la pose du système aux temps d'acquisition des données capteurs, on utilise une représentation continue de la trajectoire paramétrée en fonction du temps et deux fois dérivable : une b-spline cubique uniforme cumulée décrite par Lovegrove *et al.* [118] (introduite dans les notions de base 1.11.2). Cette représentation permet un contrôle local de la trajectoire : l'évaluation de la trajectoire en un temps t ne dépend que d'un sous-ensemble de nœuds (dans d'autres représentations comme les courbes de Béziers, cela dépend de l'intégralité des nœuds). Les nœuds de cette spline sont des points de contrôle uniformément répartis à intervalles de temps constant. Lovegrove utilise des matrices de transformation $\mathbb{R}^{4\times4}$ (*cf.* 1.1) pour représenter la pose d'un nœud d'indice i dans le repère monde :

$$\mathbf{T_i} = \begin{bmatrix} \mathbf{R_i} & \mathbf{t_i} \\ 0_{1\times3} & 1 \end{bmatrix}, \quad \mathbf{T_i} \in \mathbb{SE}3, \quad \mathbf{R_i} \in \mathbb{SO}3, \quad \mathbf{t_i} \in \mathbb{R}3, \tag{10.1}$$

avec $\mathbf{R_i}$ une matrice de rotation correspondant à l'orientation et $\mathbf{t_i}$ la position. Avec cette représentation, le déplacement relatif Δ entre deux nœuds $\mathbf{T}_1$ et $\mathbf{T}_2$ peut se calculer avec le logarithme matriciel (*cf.* 1.4) :

$$\Delta = \log(\mathbf{T_2}^{-1}\mathbf{T_1}), \quad \Delta \in \mathbb{R}^{4\times4} \tag{10.2}$$

et l'incrément sur un nœud peut se calculer avec l'exponentielle matriciel :

$$\mathbf{T}_2 = \mathbf{T}_1\exp(\Delta) \tag{10.3}$$

L'avantage de cette représentation est qu'elle unifie les opérations sur les orientations et les positions. Toutefois, le temps de calcul de l'exponentielle et du logarithme matriciel sur des matrices de dimension 4×4 est important. La formule de Rodrigues (présentée dans la section 1.4) permet de calculer l'exponentielle et le logarithme sur les matrices de rotation du groupe $\mathbb{SO}3$ avec de meilleures performances en précision et temps de calcul [1]. C'est pourquoi dans notre travail, on définit la trajectoire $\mathbf{T} = \{\mathcal{R}, \mathcal{T}\}$ composée d'une spline de position $\mathcal{T}$ et d'une spline d'orientation $\mathcal{R}$. Ainsi, le i^{ème} nœud de la trajectoire $\mathbf{T}$ est composé du i^{ème} nœud d'orientation et du i^{ème} nœud de position tels que :

$$\mathbf{T}_i = \{\mathcal{R}_i, \mathcal{T}_i\} \tag{10.4}$$

Les calculs de différences et d'incréments sont effectués avec les opérateurs + et - pour les positions de $\mathcal{T}$ et avec la formule de Rodrigues pour les orientations de $\mathcal{R}$. L'interpolation de la trajectoire au temps t, notée $\mathbf{T}(t)$, nécessite 4 points de contrôle consécutifs notés $[p_0, p_1, p_2, p_3]$ dont les temps

1. Le gain en temps de calcul est de l'ordre d'un facteur 10 pour des problèmes d'optimisation.

Position	**Orientation**
$\mathcal{T}(t) = \mathcal{T}_0 + \sum_{i=0}^{2}(\mathcal{T}_{i+1} - \mathcal{T}_i)\tilde{B}_{i+1}(u(t))$ $\dot{\mathcal{T}}(t) = \sum_{i=0}^{2}(\mathcal{T}_{i+1} - \mathcal{T}_i)\dot{\tilde{B}}_{i+1}(u(t))$ $\ddot{\mathcal{T}}(t) = \sum_{i=0}^{2}(\mathcal{T}_{i+1} - \mathcal{T}_i)\ddot{\tilde{B}}_{i+1}(u(t))$	$\mathcal{R}(t) = \mathcal{R}_0 \prod_{i=0}^{2}\exp(\tilde{B}_{i+1}(u(t))\Omega_i)$ $\dot{\mathcal{R}}(t) = \mathcal{R}_0(\dot{A}_0A_1A_2 + A_0\dot{A}_1A_2 + A_0A_1\dot{A}_2)$ $\ddot{\mathcal{R}}(t) = \mathcal{R}_0\begin{pmatrix}\ddot{A}_0A_1A_2 + A_0\ddot{A}_1A_2 + A_0A_1\ddot{A}_2 + \\ 2(\dot{A}_0\dot{A}_1A_2 + \dot{A}_0A_1\dot{A}_2 + A_0\dot{A}_1\dot{A}_2)\end{pmatrix}$

Notations		
$\Delta t = t_{i+2} - t_{i+1}, \quad u(t) = (t - t_{i+1})/\Delta t$	$A_j = \exp(\Omega_j\tilde{B}_{j+1}(u(t)))$	$\dot{A}_j = A_j\Omega_j\dot{\tilde{B}}_{j+1}(u(t))$
$\Omega_i = \log(\mathcal{R}_i^T\mathcal{R}_{i+1})$	$\ddot{A}_j = \dot{A}_j\Omega_j\dot{\tilde{B}}_{j+1}(u(t)) + A_j\Omega_j\ddot{\tilde{B}}_{j+1}$	

Fonctions de base et leurs dérivées		
$\tilde{B}_0(u) = 1.0$	$\dot{\tilde{B}}_0(u) = 0$	$\ddot{\tilde{B}}_0(u) = 0$
$\tilde{B}_1(u) = (u^3 - 3u^2 + 3u + 5)/6$	$\dot{\tilde{B}}_1(u) = (u^2/2 - u + 0.5)/\Delta t$	$\ddot{\tilde{B}}_1(u) = (u - 1)/\Delta t^2$
$\tilde{B}_2(u) = (-2u^3 + 3u^2 + 3u + 1)/6$	$\dot{\tilde{B}}_2(u) = (-u^2 + u + 0.5)/\Delta t$	$\ddot{\tilde{B}}_2(u) = (-2u + 1)/\Delta t^2$
$\tilde{B}_3(u) = u^3/6$	$\dot{\tilde{B}}_3(u) = (u^2/2)/\Delta t$	$\ddot{\tilde{B}}_3(u) = (u)/\Delta t^2$

TABLE 10.1 – Règles de calcul pour l'évaluation de la trajectoire au temps t tel que $t_1 \leq t < t_2$ en fonction des nœuds de position $\mathcal{T}_0, \mathcal{T}_1, \mathcal{T}_2, \mathcal{T}_3$ et des nœuds d'orientation $\mathcal{R}_0, \mathcal{R}_1, \mathcal{R}_2, \mathcal{R}_3$ correspondant aux temps t_0, t_1, t_2, t_3.

respectifs sont $[t_0, t_1, t_2, t_3]$. Ces points de contrôle sont choisis tels que $t_1 \leq t < t_2$. Le temps t est normalisé entre p_1 et p_2 tel que : $u = (t - t_1)/(t_2 - t_1)$. Les nœuds étant uniformément répartis en fonction du temps, $\Delta t = (t_2 - t_1)$ est constant. Ainsi, la pose au temps t s'exprime en évaluant la position et l'orientation de la trajectoire $\mathbf{T}$:

— $\mathbf{T}(t) = \{\mathcal{R}(t), \mathcal{T}(t)\}$: orientation et position,
— $\dot{\mathbf{T}}(t) = \{\dot{\mathcal{R}}(t), \dot{\mathcal{T}}(t)\}$: vitesses de rotation et de translation,
— $\ddot{\mathbf{T}}(t) = \{\ddot{\mathcal{R}}(t), \ddot{\mathcal{T}}(t)\}$: accélérations de rotation et de translation.

Les formules nécessaires à l'utilisation de cette représentation sont résumées dans le tableau 10.1.

10.3 Modèle d'évolution

En fonction du type de mouvement du système de capteurs (qui peut être monté sur une voiture, un drone ou simplement porté à la main), l'utilisation d'un modèle d'évolution permet d'appliquer une contrainte correspondante à la dynamique du système sur la trajectoire. Dans nos applications, on utilise un modèle à vitesse constante : cette contrainte minimise la somme des variations de vitesse entre chaque paire de nœuds (espacés d'un temps Δt) consécutifs de la spline $\mathbf{T}$ sur l'intervalle de temps $[t - 10, t]$ (cette contrainte s'applique sur les 10 dernières secondes de la spline) :

$$\mathcal{E}(t) = \arg\underset{\mathbf{T}}{\texttt{Min}} \sum_{t'=t-10}^{t} \left\| \frac{\dot{T}(t') - \dot{T}(t' + \Delta t)}{\sigma_{\text{evolution}}} \right\|^2 \tag{10.5}$$

Ce modèle d'évolution montre un intérêt lorsque, pour une raison quelconque, les capteurs n'envoient plus de données. De plus, il ajoute une contrainte sur les nœuds récemment ajoutés qui sont en général peu contraints.

10.4 Centrale inertielle

Une centrale inertielle est un capteur à l'image de l'oreille interne humaine. Elle mesure l'accélération et la vitesse angulaire dans un référentiel inertiel (galiléen), et l'exprime dans son repère local.

Ainsi, une centrale inertielle mesure, en plus de son mouvement propre, la rotation de la Terre et la gravité. Comme tout capteur, les données sont entachées d'erreurs, dont une partie est considérée comme du bruit gaussien, et une autre partie correspond à un biais en général constant qu'il est alors possible d'estimer. Toutefois, l'intégration des données inertielles pour obtenir une information de position multiplie les erreurs ce qui rend le résultat difficilement exploitable. C'est pourquoi, l'utilisation d'une spline deux fois dérivable permet de s'affranchir des intégrations des données inertielles. En effet, en dérivant la spline d'orientation à un temps t, on obtient une vitesse angulaire directement comparable à la donnée brute du gyromètre. On procède de manière identique avec les données accélérométriques en dérivant deux fois la spline de position. Dans les deux cas, il est nécessaire d'effectuer un changement de repère afin de se positionner dans le repère de la centrale inertielle. Ainsi, les erreurs sont estimées dans l'espace des données. En tenant compte du biais, le calcul de l'erreur entre le gyromètre et la spline sur l'intervalle de temps $[t-10, t]$ est exprimé par :

$$\mathcal{E}(t) = \arg\underset{\mathbf{T}}{\texttt{Min}} \sum_{t'=t-10}^{t} \left\| \frac{\text{Gyro}_{t'} - \mathcal{R}^T(t').\dot{\mathcal{R}}(t') + \text{Bias}_{\text{Gyro}}}{\sigma_{Gyro}} \right\|^2 \tag{10.6}$$

Et le calcul de l'erreur entre l'accéléromètre et la spline sur l'intervalle de temps $[t-10, t]$ est exprimé par :

$$\mathcal{E}(t) = \arg\underset{\mathbf{T}}{\texttt{Min}} \sum_{t'=t-10}^{t} \left\| \frac{\text{Accel}_{t'} - \mathcal{R}^T(t').(\ddot{\mathcal{T}}(t') + g) + \text{Bias}_{\text{Accel}}}{\sigma_{Accel}} \right\|^2 \tag{10.7}$$

avec g le vecteur de gravité dans le repère monde fixé comme une constante du problème. En effet, il est possible de supprimer la gravité de l'accéléromètre (ce qui nécessite de connaître l'orientation), ou d'estimer ce vecteur avec les autres paramètres du problème. Toutefois, en considérant le vecteur gravité comme une constante fixée à $(0, 0, 9.81) m.s^{-2}$ ou préalablement étalonnée, on impose une contrainte forte sur l'orientation du système tout en limitant le nombre de *ddl.* à optimiser.

10.5 GPS

Le GPS (*Global Positioning System*) est un capteur de référence dans le domaine de la localisation en extérieur. Son utilisation nécessite une constellation de satellites (au minimum 4) synchronisés qui transmettent en continu leurs positions datées. Celles-ci sont utilisées pour trianguler la position d'un récepteur GPS présent à la surface du globe. Ainsi, le GPS apporte uniquement une information de position. L'orientation de déplacement peut être déduite *a posteriori* en analysant plusieurs positions successives ou en utilisant en complément une centrale inertielle. Dans le cas d'applications professionnelles, il est courant d'utiliser un GPS dit RTK pour *Real Time Kinematic*. Cela consiste à corriger la triangulation en utilisant une base placée à proximité et qui est précisément géo-référencée. L'utilisation d'un GPS-RTK permet une localisation centimétrique.

L'utilisation des données GPS notées t_{GPS} au temps t pour appliquer une contrainte sur la spline est réalisée de manière triviale en minimisant la distance euclidienne entre la position GPS et la spline sur l'intervalle de temps $[t-10, t]$:

$$\mathcal{E}(t) = \arg\underset{\mathbf{T}}{\texttt{Min}} \sum_{t'=t-10}^{t} \left\| \frac{t'_{GPS} - \mathcal{T}(t')}{\sigma_{GPS}} \right\|^2 \tag{10.8}$$

10.6 Camera

Les approches de localisation par vision offrent une excellente précision sur le déplacement relatif à la fois sur la position que sur l'orientation. La précision du déplacement absolue est relativement correcte. Toutefois, celle-ci est connue pour dériver sur de longues distances.

Le résultat de la localisation est utilisé pour contraindre la position et l'orientation de la spline à un temps donné. De plus, la spline peut également limiter la dérive de la localisation en apportant une contrainte supplémentaire dans le processus de reconstruction 3D. C'est pourquoi, les contraintes présentées ici sont utilisées pour optimiser à la fois la spline et les images clés. Deux types de contraintes ont été développées et sont présentées ci-dessous : des contraintes en pose absolue, et des contraintes en pose relative.

10.6.1 Contrainte de pose absolue

La contrainte de pose absolue s'applique assez simplement. Il s'agit de minimiser la différence entre la pose caméra $\{\mathbf{R}, \mathbf{t}\}$ et la spline au temps de la pose caméra $\{\mathcal{R}(t), \mathcal{T}(t)\}$. Pour exprimer l'erreur angulaire entre les poses consécutives, on utilise le logarithme matriciel sur la différence des poses. La contrainte minimise la somme des erreurs angulaires sur l'intervalle de temps $[t-10, t]$:

$$\mathcal{E}(u) = \underset{\mathcal{T},\mathbf{t}}{\arg\texttt{Min}} \sum_{t'=t-10}^{t} \left\| \frac{\mathcal{T}(t') - \mathbf{t}}{\sigma_{\mathbf{t}}} \right\|^2 + \sum_{t'=t-10}^{t} \underset{\mathcal{R},\mathbf{R}}{\arg\texttt{Min}} \left\| \frac{(\log(\mathcal{R}(t')^T \mathbf{R}))}{\sigma_{\mathbf{R}}} \right\|^2 \tag{10.9}$$

Cette contrainte est minimisée en optimisant à la fois les paramètres de la spline et les paramètres de la pose caméra. La spline agit comme une contrainte sur les poses caméra.

10.6.2 Contrainte de la spline sur la caméra

En fonction des contraintes appliquées à la spline, celle-ci ne passera pas exactement sur les poses des images clés. Ainsi, il est possible qu'une contrainte GPS fasse évoluer la spline en parallèle (avec un décalage latéral). Dans une telle configuration, la contrainte absolue provoque une discontinuité sur la trajectoire caméra et altère la reconstruction 3D. C'est pourquoi, il est préférable d'utiliser une contrainte en déplacement relatif[2] entre la spline et la caméra. Considérons deux images clés consécutives $\{\mathbf{R_1}, \mathbf{t_1}\}$ et $\{\mathbf{R_2}, \mathbf{t_2}\}$ avec comme temps respectifs t_1 et t_2. Cette contrainte impose un déplacement entre les deux images clés correspondant au déplacement relatif de la spline entre les temps t_1 et t_2 noté $\Delta(\mathbf{T_1}, \mathbf{T_2})$ et qui est obtenu par changement de repère :

$$\Delta(\mathbf{T_1}, \mathbf{T_2}) = \{\mathcal{R}(t_2)^T \mathcal{R}(t_1) \ , \ \mathcal{R}(t_1)^T (\mathcal{T}(t_2) - \mathcal{T}(t_1))\} \tag{10.10}$$

Le déplacement entre les deux images clés est obtenu ainsi :

$$\Delta(\{\mathbf{R_1}, \mathbf{t_1}\}, \{\mathbf{R_2}, \mathbf{t_2}\}) = \{\mathbf{R_2}^T \mathbf{R_1} \ , \ \mathbf{R_1}^T (\mathbf{t_2} - \mathbf{t_1})\} \tag{10.11}$$

La contrainte de pose relative minimise la somme des différences entre le déplacement $\Delta(\mathbf{T_1}, \mathbf{T_2})$ de la spline et le déplacement entre les paires consécutives d'images clés $\Delta(\{\mathbf{R_1}, \mathbf{t_1}\}, \{\mathbf{R_2}, \mathbf{t_2}\})$ sur l'intervalle de temps $[t-10, t]$ avec $\Delta t_{12} = t_2 - t_1$:

$$\mathcal{E}(t) = \underset{\mathbf{T},\mathbf{R_1},\mathbf{t_1},\mathbf{R_2},\mathbf{t_2}}{\arg\texttt{Min}} \sum_{t'=t-10}^{t} \left\| \frac{\mathcal{R}(t')^T (\mathcal{T}(t' + \Delta t_{12}) - \mathcal{T}(t')) - \mathbf{R_1}^T (\mathbf{t_2} - \mathbf{t_1})}{\sigma_{\mathbf{t}}} \right\|^2 + \underset{\mathbf{T},\mathbf{R_1},\mathbf{R_2}}{\arg\texttt{Min}} \sum_{t'=t-10}^{t} \left\| \frac{\log((\mathcal{R}(t' + \Delta t_{12})^T \mathcal{R}(t'))^T (\mathbf{R_2}^T \mathbf{R_2}))}{\sigma_{\mathbf{R}}} \right\|^2 \tag{10.12}$$

2. Le déplacement relatif entre les poses T_1 et T_2 correspond à la pose T_2 exprimée dans le repère de T_1.

10.7 Optimisation incrémentale de la trajectoire

Pour fusionner les contraintes de mouvement, la trajectoire du système est représentée par une spline construite à partir des données capteurs (GPS, IMU, caméra, ...). Celle-ci est également utilisée pour propager les contraintes, entre un GPS et les images clés de la caméra par exemple. C'est pourquoi, on considère la spline comme une contrainte supplémentaire qui est ajoutée au graphe de dépendances. Ainsi, les points de contrôle de la spline sont ajoutés aux processus d'optimisations avec le reste des paramètres du graphe. La sélection incrémentale des paramètres à optimiser s'effectue pendant le parcours du graphe de dépendances avant chaque optimisation.

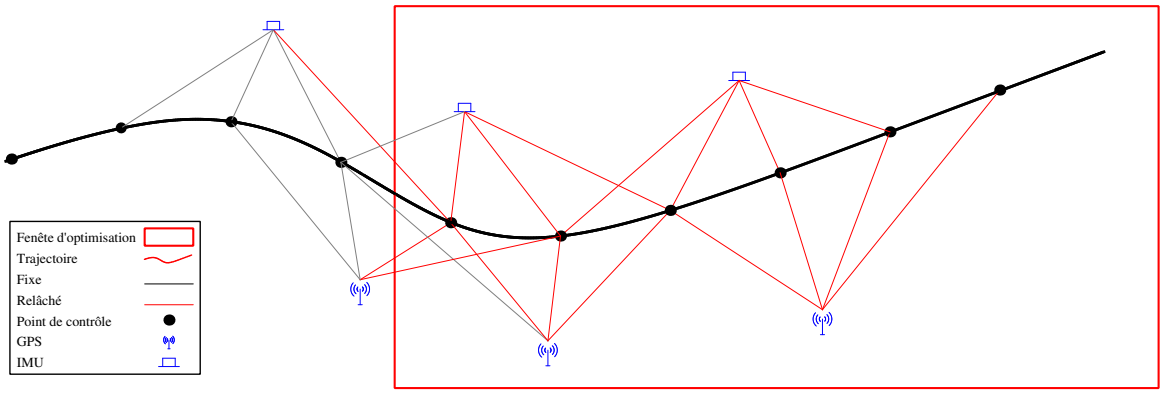
10.7.1 Sélection incrémentale des paramètres à optimiser

En présence de la spline dans le processus d'optimisation, il convient de redéfinir la sélection des paramètres à optimiser car les nœuds de la spline sont créés à intervalle de temps régulier même en l'absence de mouvement. Ainsi, les n dernières images clés peuvent correspondre à un nombre très important de nœuds de la spline. C'est pourquoi, les nœuds optimisés dans le processus incrémental sont sélectionnés avec une fenêtre temporelle glissante correspondant aux 10 dernières secondes (cette durée est ajustable en fonction des applications et du temps séparant chaque nœud). Dans cet intervalle de temps, les nœuds sont optimisés (en position et orientation) en utilisant toutes les contraintes de mouvement. Cependant, il est nécessaire de gérer la transition entre « nœud optimisé » et « nœud non-optimisé » en utilisant le schéma de sélection incrémentale décrit par Steedly *et al.* [62] pour la sélection des contraintes. Dans le cas contraire, les nœuds optimisés peu contraints feront diverger la spline juste avant de sortir de la fenêtre d'optimisation. Les associations entre les nœuds de la spline et les contraintes sont illustrées par la figure 10.2. L'ensemble des paramètres, de nœuds, poses caméra et points 3D, associé aux observations incluses dans l'intervalle de temps $[t-10, t]$ est optimisé. De plus, les observations précédant l'intervalle de temps $[t-10, t]$, donc à l'extérieur de la fenêtre d'optimisation (*cf.* la figure 10.2a), sont également incluses dans la fonction de coût si elles sont liées à un paramètre à l'intérieur de l'intervalle optimisé. Les erreurs de reprojection sont sélectionnées de façon similaire comme le montre la figure 10.2b : on sélectionne tous les points 3D présents dans la fenêtre d'optimisation ainsi que les observations dans les caméras qui ne sont plus optimisées. Il s'agit là du schéma classique utilisé pour l'ajustement de faisceaux incrémental qui inclut dans la fonction de coût des observations associées à des poses non optimisées pour fixer la trajectoire. Cela produit un ancrage des paramètres optimisés pour conserver une certaine cohérence avec les paramètres plus anciens. Par conséquence, les contraintes sont appliquées uniquement aux paramètres correspondant à la fin de la reconstruction (spatialement et temporellement). En pratique, le nœud de paramètre correspondant à la spline est le premier à être visité lors du parcours du graphe (ceci est définit par les dépendances).

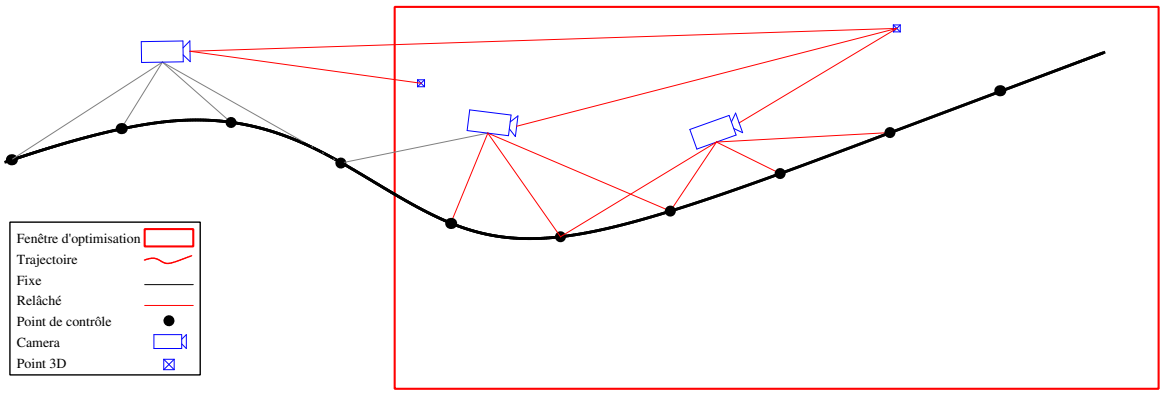
10.8 Localisation

Dans le MCSLAM, la localisation est réalisée avec une optimisation des paramètres de la trajectoire au temps courant t. Tout comme le processus d'optimisation incrémentale, l'algorithme de localisation est géré avec un graphe de dépendances. Cela permet de faciliter la gestion des différentes contraintes. Toutefois, ce graphe de dépendances est limité aux paramètres de trajectoire. Aucune mise à jour de la carte 3D n'est effectuée par exemple.

Idéalement, le résultat de la localisation est donné par l'évaluation de la spline au temps courant t. Cette évaluation nécessite 4 points de contrôle, deux avant et deux après t. Or, un problème se pose. A l'instant t, il n'est pas possible de positionner précisément les 2 nœuds à venir, donc d'évaluer avec précision la pose courante. Une solution consiste à retarder l'évaluation de la spline : en considérant Δt le temps entre deux nœuds, on pourrait redéfinir le temps courant tel que $t' = t - 2 \times \Delta t$. Avec ce



(a) Sélection incrémentale des contraintes GPS et inertielles sur la spline.



(b) Sélection incrémentale des contraintes de pose sur la spline et d'erreurs de reprojection.

FIGURE 10.2 – Sélection incrémentale des paramètres et des contraintes. La figure 10.2a montre que les noeuds sont également contraints par des observations à l'extérieur de la fenêtre d'optimisation. De plus, certaines contraintes sont associées à moins de 4 nœuds optimisés. La figure 10.2b montre que la même fenêtre d'optimisation est utilisée pour sélectionner les erreurs de reprojection entre les points 3D et les poses caméra. Seules les poses caméra, les points 3D et les noeuds à l'intérieur de la fenêtre sont optimisés.

retard, les 4 nœuds nécessaires à l'évaluation de la spline sont à la fois connus et contraints. Ce retard semble inévitable avec la modélisation choisie.

Toutefois, les applications visées (visualisation temps réel pour la Réalité Augmentée et guidage pour la navigation autonome) imposent des contraintes de localisation temps réel. C'est pourquoi, on utilise en pratique comme résultat de localisation la solution du calcul de pose caméra sur l'image acquise au temps t. La spline influe indirectement sur ce résultat car le calcul est effectué à partir des points 3D optimisés conjointement avec la spline.

10.9 Création des nouveaux nœuds

La création des nouveaux nœuds se révèle compliquée. Deux stratégies sont envisageables. La première consiste à placer les nouveaux nœuds en utilisant un modèle de prédiction (à vitesse constante par exemple). Toutefois, le nouveau nœud sera peu contraint et provoquera des irrégularités au niveau de la « tête » de la spline. Le modèle d'évolution tend à limiter cela. Une autre solution est de retarder la création du nœud jusqu'à avoir une bonne estimation de la pose du système au temps du nœud (provenant d'un GPS ou d'une caméra). Cette solution a le désavantage de retarder l'utilisation des contraintes sur la spline : il est nécessaire que les données capteurs dépassent le dernier nœud pour effectuer un calcul de pose et positionner le nouveau nœud. Dans un tel processus, la spline se révèle peu utile. En pratique, les applications visées (que ce soit la réalité augmentée ou la navigation autonome) nécessitent une réponse rapide de l'algorithme de localisation. C'est pourquoi, on utilise la première solution consistant à prédire la pose des nœuds.

10.10 Conclusion

L'utilisation d'une spline temporelle permet l'utilisation de contraintes asynchrones de natures différentes pour estimer une même trajectoire. De plus, l'approche est générique sur le type de trajectoires. En effet, l'élément important est l'utilisation d'une trajectoire continue. En pratique, la modélisation choisie se révèle difficile à utiliser dans un processus incrémental, car évaluer la spline en un temps nécessite 4 points de contrôle. En supposant que les nœuds sont espacés d'une seconde ($\Delta t = 1$), il faut acquérir des données capteurs pendant $3 \times \Delta t$ soit 3 secondes avant de pouvoir commencer à utiliser la spline. Pour la même raison, utiliser la spline pour estimer la pose à un temps t' n'est possible que si le temps courant est supérieur à $t' + 2 \times \Delta t$, ce qui correspond à un retard de $2 \times \Delta t$. C'est pourquoi, il semble nécessaire d'utiliser cette représentation en complément d'un algorithme de SLAM classique, comme une contrainte supplémentaire du problème, ce que le MCSLAM permet de faire.

Chapitre 11

Expérimentations

Ce chapitre présente une série d'expérimentations sur des applications réelles de SLAM contraint. L'objectif est de mettre en évidence les apports, en matière de précision, liés à l'utilisation de contraintes supplémentaires dans les méthodes de SLAM. De plus, toutes les variantes de SLAM testées sont générées avec le MCSLAM (y compris le SLAM utilisant uniquement la contrainte de reprojection). En plus de montrer la modularité du *framework*, cela permet de tester différentes configurations dont les seules différences sont les contraintes utilisées. C'est pourquoi, si un algorithme marche mieux lorsqu'on ajoute une contrainte, on peut déduire que cette différence est une conséquence de la contrainte (ce qui serait plus difficile à montrer en utilisant des implémentations utilisant des descripteurs différents par exemple). Deux types d'expérimentations sont présentés : le MCSLAM avec des contraintes de structure dans la section 11.1 et avec des contraintes de mouvement dans la section 11.2.

11.1 MCSLAM avec contraintes de structure

Cette première série d'expérimentations implique des contraintes de structure. Celles-ci correspondent à des objets 3D dont la forme et la position dans l'environnement sont approximativement connues. Aucune connaissance *a priori* sur l'apparence des objets n'est utilisée. L'objectif est de proposer une solution précise et performante pour des applications de Réalité Augmentée avec recalage simultané d'objets multiples, statiques et rigides étant initialisés en ligne et approximativement connus. Pour effectuer le recalage entre les modèles et les objets observés dans l'image, on utilise le MCSLAM avec les contraintes relatives aux objets détaillées dans le chapitre 9. Comme les applications visées concernent la Réalité Augmentée, les critères d'évaluation des méthodes sont la qualité du recalage dans l'image et le temps de calcul. La dérive du SLAM est dans une certaine mesure négligée tant que le recalage est correctement effectué. Le capteur utilisé est une caméra vidéo *global shutter* cadencée à 60 images par seconde avec une résolution de 640×480 au format *bayer*[1]. On utilise un ordinateur de bureau équipé d'un processeur i7 3.4GHz et deux fils d'exécution sont créés : un pour la détection, l'appariement et la localisation ; le second pour la mise à jour de la carte et l'optimisation.

Les premières expérimentations sont effectuées sur des objets basiques pour valider le MCSLAM qualitativement et quantitativement. Puis des expérimentations avec des objets plus complexes et des contraintes d'amers visuels supplémentaires (segments 3D) seront présentées. Trois configurations du MCSLAM sont testées.

1. SLAM : un SLAM classique qui n'utilise aucune contrainte liée aux objets 3D présents.
2. $\overline{\text{MCSLAM}}$: un SLAM contraint par les objets (*i.e.* les points 3D du SLAM sont contraints à se rapprocher de la surface des objets), mais les paramètres des objets ne sont pas optimisés. Cette configuration correspond à la meilleure approche actuelle de l'état de l'art, et offre une

1. Image couleur codée sur une matrice de $640 \times 480 \times 8$ bits donc avec un seul canal.

très bonne précision lorsqu'elle est utilisée avec un unique objet observé dès le début du SLAM (*cf.* état de l'art sur les travaux de Tamaazousti *et al.* [4]).

3. MCSLAM : un SLAM contraint par des objets qui sont également optimisés pour satisfaire les contraintes.

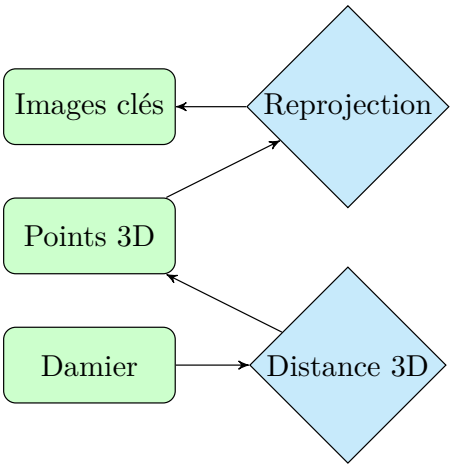
Ces premières expérimentations n'impliquent pas de contraintes de mouvement donc l'intérêt d'utiliser une trajectoire continue est limitée. C'est pourquoi, dans cette section la trajectoire est modélisée par des images clés.

11.1.1 Application au recalage d'un objet connu : le damier

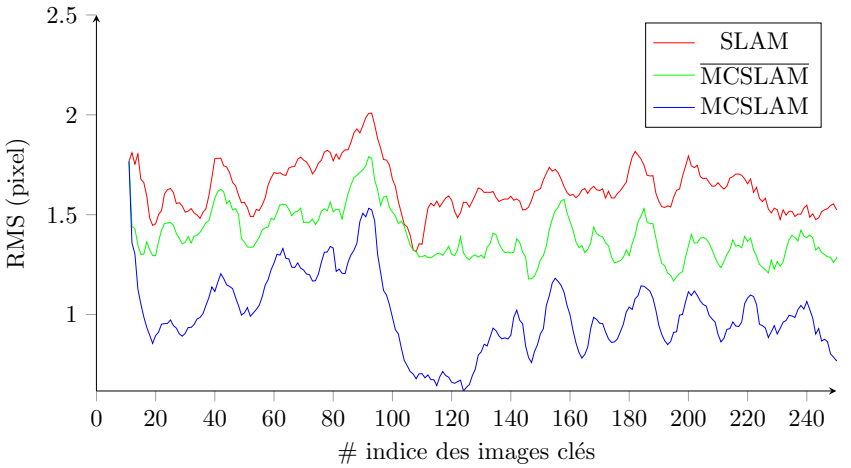
Pour ce premier scénario, l'objet utilisé est un damier de calibrage composé de 9×7 cases. L'intérêt de cet objet est qu'il est simple à modéliser. De plus il constitue une vérité terrain car le modèle est connu. On suppose que le détecteur de coin (Harris) ne trouvera que des points d'intérêt à l'intersection des cases. C'est pourquoi, on génère un damier virtuel de dimension 9×7 mètres (unité arbitraire à un facteur d'échelle près) et composé de 10×8 points d'intersections de cases (les cases sont carrées). La contrainte 3D consiste alors à minimiser la distance entre les points 3D du SLAM et les points 3D du damier (une étape d'association au point le plus proche est nécessaire). La méthode MCSLAM optimise les 7 *ddl.* de l'objet : 6 pour la pose et 1 pour le facteur d'échelle.

L'expérimentation se compose d'une vidéo de 2 minutes et approximativement 250 images clés sont utilisées pour chaque méthode. Pour chacune des méthodes testées, le modèle de damier (pose et facteur d'échelle) est initialisé exactement dans les mêmes conditions : les méthodes de SLAM sont exécutées en observant le damier et après la 10^e image clé, une ICP [2] est réalisée entre les points 3D du SLAM (sélectionnés manuellement en délimitant avec des clics souris la position du damier dans l'image) et les points 3D du modèle de damier. La figure 11.1b montre l'évolution, à chaque image clé, de l'erreur entre les points d'intérêt détectés et l'erreur de reprojection dans l'image des points 3D du modèle de damier. Les évolutions de l'erreur sont liées aux mouvements de la caméra autour de l'objet et les erreurs les plus importantes sont obtenues lorsque la caméra est proche du motif. On observe que les RMS [3] sont identiques à l'initialisation du damier (à la 10^e image clé). Puis, l'erreur de la méthode MCSLAM décroit rapidement car la pose de l'objet et les points 3D sont optimisés pour minimiser la contrainte 3D. L'erreur de la méthode $\overline{\text{MCSLAM}}$ décroit légèrement car seuls les points 3D sont optimisés. La figure 11.2 illustre la qualité du recalage du modèle de damier dans l'image selon les 3 méthodes à deux instants de la séquence, l'image clé 94 correspondant à l'erreur de recalage la plus importante pour les 3 méthodes, et l'image clé 124 qui correspond à l'erreur la moins importante pour la méthode MCSLAM.

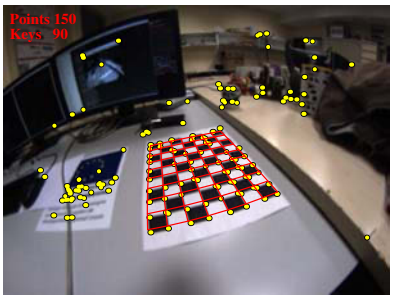
2. Iterative closest point : méthode itérative pour estimer le recalage entre 2 nuages de points 3D.
3. *Root Mean Square* : racine de la somme des erreurs au carré.



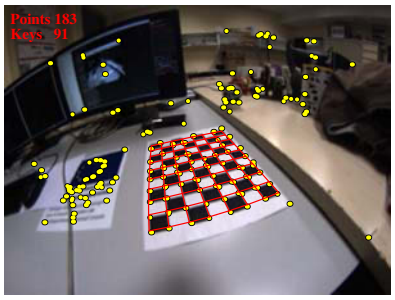
(a) Graphe du MCSLAM



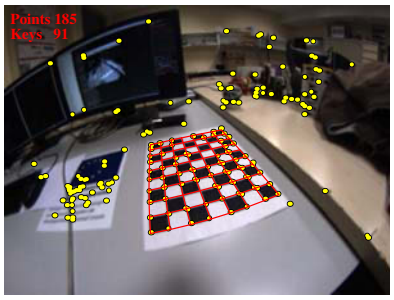
(b) Évolution de l'erreur 2D de recalage avec le modèle de damier et les points d'intérêt détectés aux intersections des cases dans l'image.



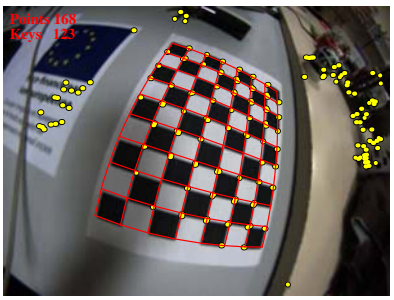
(c) SLAM

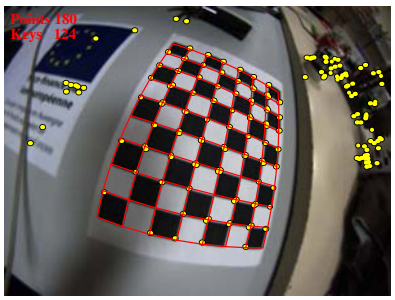


(d) $\overline{\text{MCSLAM}}$



(e) MCSLAM





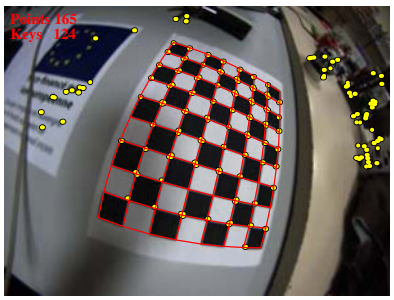


FIGURE 11.1 – Comparaisons visuelles entre le SLAM, le $\overline{\text{MCSLAM}}$ et le MCSLAM. Le numéro de l'image clé auquel les images sont enregistrées est noté dans le coin supérieur gauche après *Keys* (*cf.* la figure 11.1b pour voir l'erreur de recalage correspondante).

11.1.2 Application au recalage d'objets 3D de formes basiques

Ce second scénario implique 4 objets 3D dont les dimensions réelles ne sont pas connues : un plan, une sphère, un parallélépipède rectangle et un cylindre. L'objectif de l'expérimentation est d'évaluer la sensibilité des méthodes face aux initialisations grossières d'objets. L'expérimentation, illustrée par la figure 11.2, compare les méthodes SLAM, $\overline{\text{MCSLAM}}$ et MCSLAM sur une application de réalité augmentée où des modèles d'objets sont projetés dans les images. Pour de telles applications, le critère évalué est la qualité visuelle du recalage entre les modèles projetés dans l'image et les objets présents. Chaque méthode est évaluée sur la même séquence vidéo et les objets sont initialisés de manière identique.

On remarque que l'erreur de recalage des objets avec le SLAM 11.2b est importante car aucune contrainte ne vient limiter l'accumulation des erreurs. L'approche $\overline{\text{MCSLAM}}$ 11.2c montre un résultat meilleur que l'approche SLAM, mais l'erreur de recalage des objets reste trop importante pour des applications de Réalité Augmentée. Si un seul objet était visible, la contrainte associée aurait modifié la trajectoire de la caméra et le recalage aurait été correct. Mais dans une configuration à plusieurs objets, il n'est pas possible de corriger la trajectoire pour satisfaire l'ensemble des contraintes objets. C'est pourquoi il est nécessaire de libérer les degrés de liberté de chaque objet afin que la reconstruction globale soit meilleure. C'est ce qui est fait avec l'approche MCSLAM 11.2d, dont les objets étaient aussi mal initialisés que pour les autres approches, mais l'optimisation des objets a permis une meilleure reconstruction et l'ensemble des objets se projette correctement dans l'image.

11.1.3 Application au recalage d'objets 3D de formes complexes

Ce dernier scénario illustre l'utilisation du MCSLAM avec à la fois des objets simples et des objets de formes complexes (préalablement photo-modélisés). Les amers 3D sont de types points 3D et segments 3D [4] reconstruits en ligne et utilisés comme contraintes sur les objets 3D. Tout comme les points 3D, les segments 3D sont utilisés à la fois pour reconstruire l'environnement (et donc localiser la caméra en utilisant les points de contours dans les images) et comme contrainte sur les objets 3D. Toutefois, il est difficile de contraindre des segments 3D à la surface des objets courbes. Ce problème n'est pas abordé ici mais le lecteur intéressé pourra se référer aux travaux de Loesch *et al.* [134]. C'est pourquoi, les segments 3D ne sont utilisés que pour contraindre les objets composés de plans : objets plan et parallélépipède rectangle (mais rien n'empêche l'algorithme de SLAM d'utiliser des segments qui sont sur des objets complexes pour la contrainte d'erreur de reprojection).

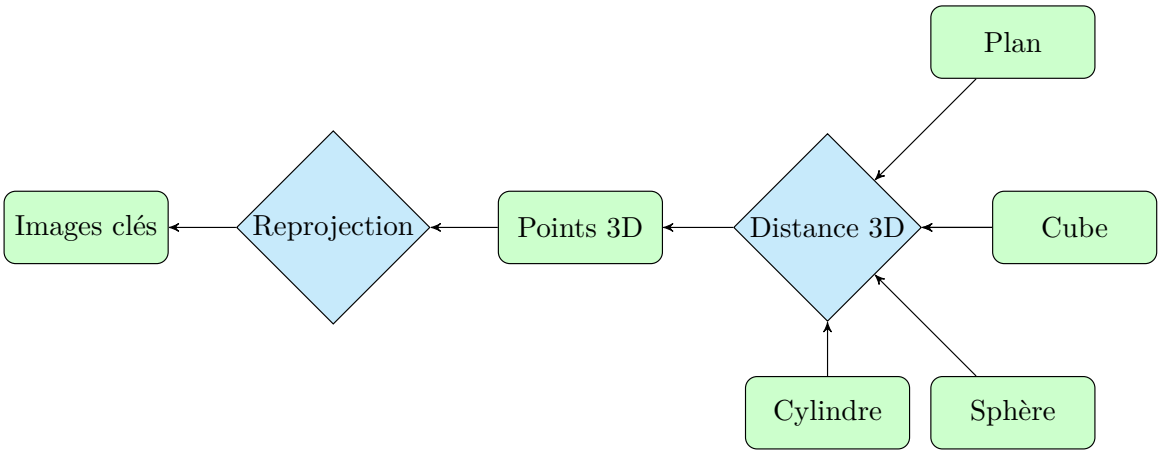
Le résultat de l'expérimentation est visible sur la figure 11.3. La séquence est composée de $10K$ images, le SLAM fait intervenir des points et des segments 3D, et 8 objets de 5 classes différentes (plan, sphère, parallélépipède rectangle, cylindre et modèle 3D). Tous les objets sont visibles dans les images et utilisés pour contraindre la reconstruction. Pour cette application, les performances du MCSLAM sont temps réel : le processus de localisation nécessite en moyenne 6 ms ce qui est suffisant pour une acquisition à 60 images par seconde. De plus, deux processus sont utilisés, un pour la localisation et un pour la mise à jour de la carte :

1. localisation : en moyenne, 5 ms sont nécessaires pour la détection d'amers 2D (Harris et Canny [5]) et l'appariement, 1 ms pour le calcul de pose
2. mise à jour : en moyenne, 70 ms sont nécessaires pour la mise à jour de carte. Ce temps est significatif à cause des calculs de dérivées effectués numériquement pour la reconstruction des segments 3D ainsi que les contraintes entre segments et objets.

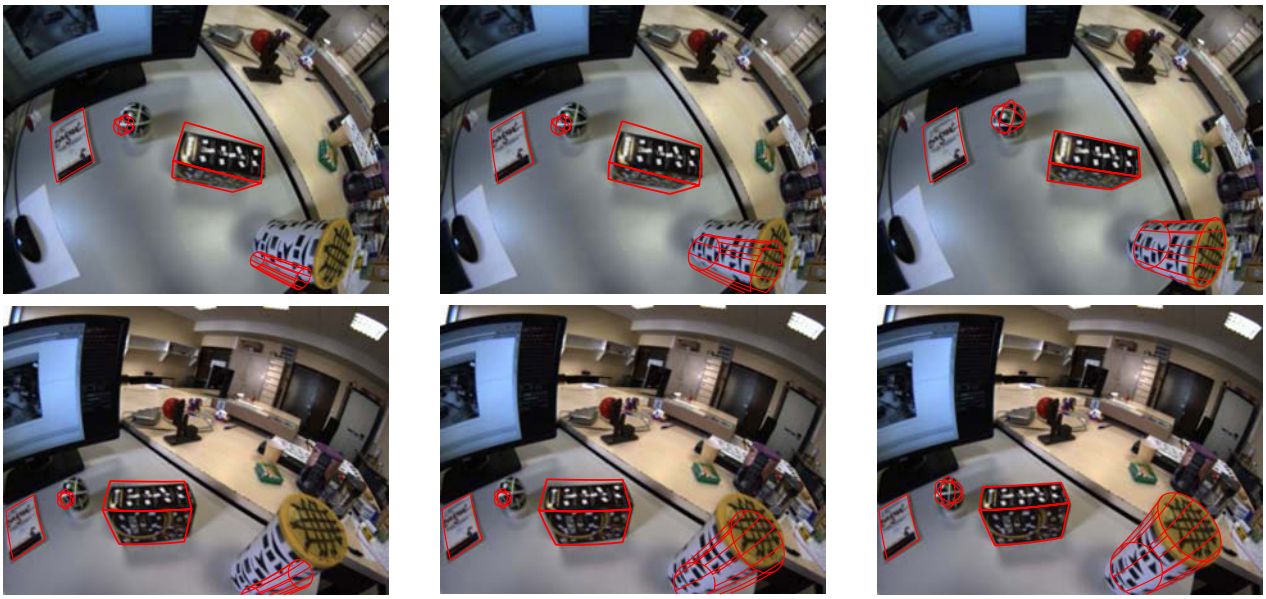
La mise à jour de la carte dans un processus en parallèle permet au processus de localisation de traiter toutes les images. Dans le cas contraire, chaque optimisation bloquerait la localisation pendant 70 ms

4. la gestion (détection, extraction et mise en correspondance) et la paramétrisation des segments 3D sont celles décrites par Klein *et al.* [85]

5. Algorithme de détection de contours. L'implémentation utilisée est celle disponible dans OpenCV.

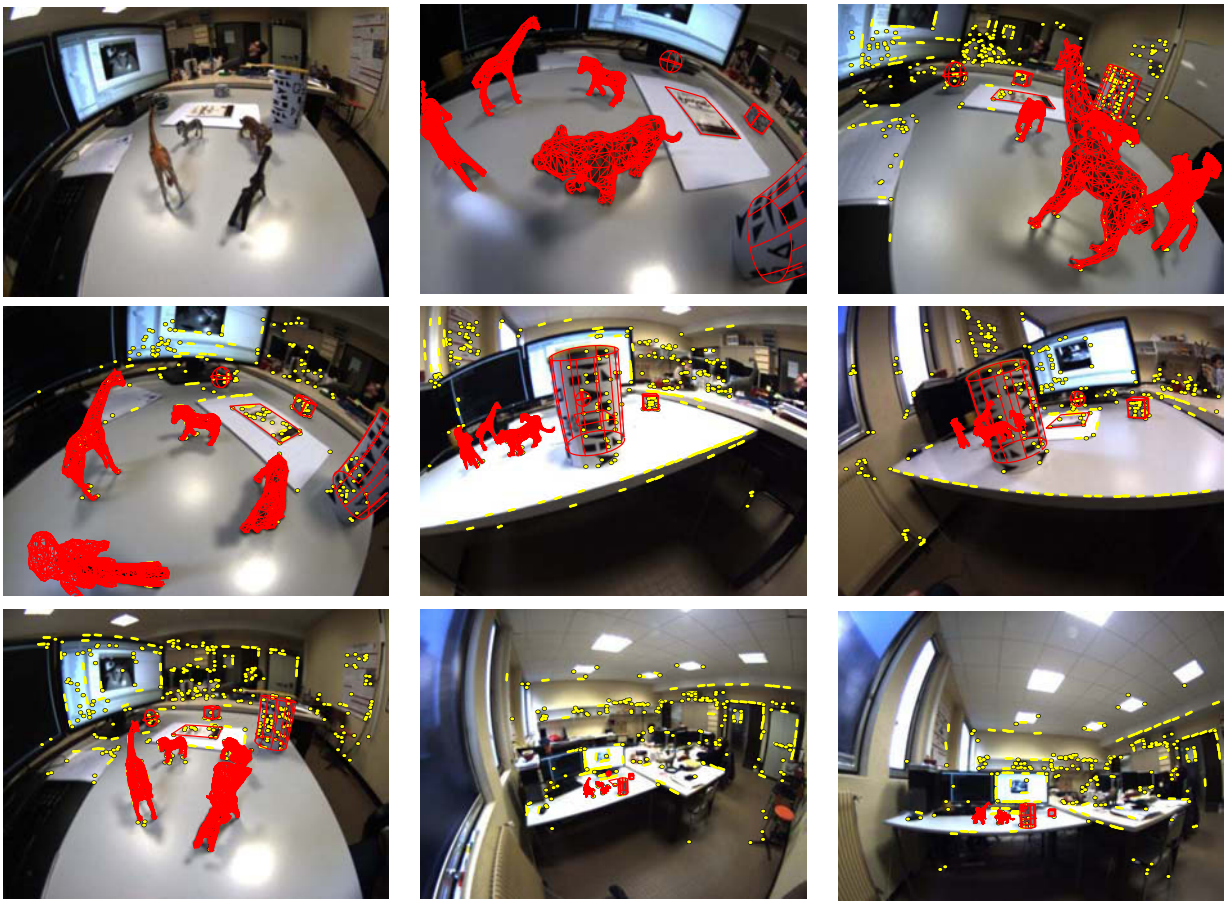


(a) Graphe

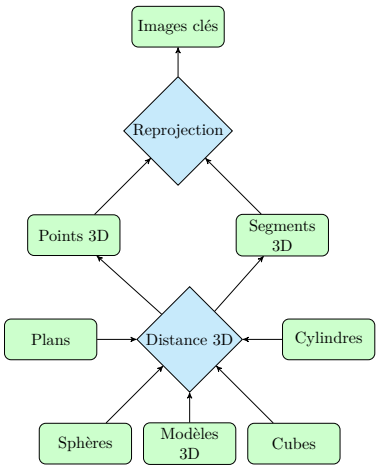


(b) SLAM (c) $\overline{\text{MCSLAM}}$ (d) MCSLAM

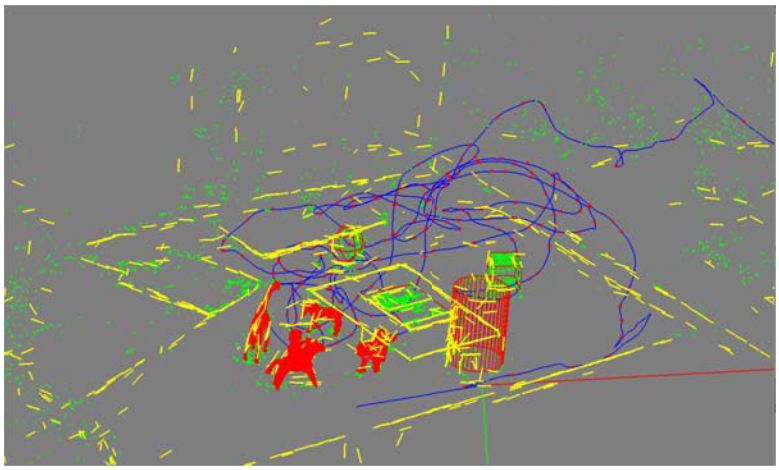
FIGURE 11.2 – Application à la Réalité Augmentée : comparaison visuelle sur le recalage d'objets entre les méthodes SLAM, $\overline{\text{MCSLAM}}$ et MCSLAM.



(a) Résultat du recalage entre les modèles d'objets projetés dans l'image (en rouge) et les objets réels. Les points et les traits jaunes correspondent aux points et contours détectés avec les détecteurs Harris et Canny.



(b) Graphe



(c) Vue 3D de la reconstruction : les points verts correspondent aux points 3D, les traits jaunes aux segments 3D, les objets 3D sont en rouges, la ligne bleue correspond à la trajectoire de la caméra et les points rouges aux images clés.

FIGURE 11.3 – Illustration du MCSLAM pour le recalage d'objets en utilisant des points et des segments 3D.

ce qui peut nuire à la fois à la qualité du calcul de pose mais également au rendu visuel qui serait moins fluide.

11.2 MCSLAM avec contraintes de mouvement

Cette seconde série d'expérimentations implique des contraintes de mouvement. Celles-ci correspondent aux données de position et d'inertie (vitesse de rotation et accélération) fournies par un GPS et une centrale inertielle rigidement liés à la caméra. L'objectif est d'évaluer l'utilisation d'une représentation continue de la trajectoire pour fusionner les différents capteurs dans le cadre de la navigation pour la robotique mobile. Cela est réalisé avec le MCSLAM pour construire la trajectoire modélisée par une spline comme expliqué dans le chapitre 10. Comme les applications visées concernent la navigation de robot autonome, les critères d'évaluation sont la précision de la méthode et le temps de calcul.

Les données utilisées proviennent de la base KITTI [70]. Cette base contient un nombre important de trajectoires, composées des enregistrements de 4 caméras vidéos, d'un GPS-RTK [6] couplé à une centrale inertielle [7], et d'un vélodyne [8] qui ne sera pas utilisé. Pour chaque caméra, la base de données contient à la fois les images brutes et les images corrigées de la distorsion. Par simplicité, on utilise les images corrigées de la caméra *global shutter* référencée « image_00 » de résolution 1226×370. De plus, on utilise les données GPS et inertielles brutes et asynchrones avec les caméras : l'acquisition de la centrale inertielle et du GPS est effectuée à 100Hz et les caméras enregistrent à 10Hz. Des données inertielles synchronisées *a posteriori* avec les caméras sont disponibles dans la base KITTI mais ne seront pas utilisées.

La première expérimentation a pour but de valider la contrainte en déplacement relatif présentée dans la section 10.6, c'est pourquoi on utilise le GPS qui envoie des données fiables. Les expérimentations suivantes présentent le résultat du MCSLAM sur des jeux de données de la base KITTI en utilisant uniquement la caméra et la centrale inertielle. Les dernières expérimentations illustrent l'utilisation du MCSLAM afin d'effectuer du calibrage hors-ligne avec la contrainte GPS. Les jeux de données sont sélectionnés pour leurs grandes dimensions. Leurs caractéristiques sont résumées dans le tableau 11.1.

Dans toutes les expérimentations qui suivent, l'intervalle de temps entre deux nœuds de spline est fixé arbitrairement à $\Delta t = 0.5$ seconde. La fenêtre glissante de sélection des paramètres optimisés (*cf.* 10.7.1) est de 10 secondes. Les pondérations de chaque contrainte sont identiques pour l'ensemble des expérimentations et fixées empiriquement afin qu'aucune ne soit marginalisée dans le processus d'optimisation. Le détecteur de points d'intérêt FAST est utilisé et asservi pour extraire approximativement 600 points par image. Le critère de sélection des images clés sélectionne en moyenne 1 image clé toutes les 3 images. Le nombre maximum d'images clés optimisées dans le SLAM est fixé à 20.

La précision des méthodes de localisation est évaluée en utilisant comme vérité terrain une spline optimisée en position et orientation sur les données GPS-RTK. L'utilisation d'une spline (dont les nœuds sont espacés d'un temps $\Delta t = 0.5$ seconde) sur les données GPS permet d'interpoler la vérité terrain aux temps de prise des images (asynchrones avec le GPS). L'initialisation du SLAM est contrainte à suivre la vérité terrain sur les 10 premières images clés afin d'initialiser le SLAM dans le même repère que le GPS. L'intérêt est de faciliter la comparaison avec la vérité terrain. Étant donné qu'il n'est pas possible d'évaluer la spline au temps courant (comme expliqué section 10.8), les erreurs sont mesurées sur le résultat de la localisation par vision à chaque image. Deux types d'erreurs sont mesurées :

6. haute résolution
7. la précision donnée par le constructeur est de 0.02m/0.1°.
8. capteur tournant composés de 64 nappes de lasers.

	09_30_18	09_30_28	10_03_34	10_03_27
Durée	4 min 47	8 min 57	8 min 3	7 min 50
Kilomètres	3.3	6.3	7.7	5.6
# images	2764	5179	4665	5650
# GPS/IMU	28670	54663	47945	45700

TABLE 11.1 – Caractéristiques des jeux de données sélectionnés dans la base KITTI.

1. erreur absolue : distance euclidienne entre la trajectoire reconstruite et la vérité terrain au temps des images,
2. erreur relative : différence entre la distance relative entre les images clés consécutives datées t_i et t_{i+1} et le déplacement relatif dans la vérité terrain aux temps t_i et t_{i+1}.

Le tableau 11.2, à la fin de la section, résume les expérimentations avec les résultats sous forme d'erreurs moyennes et d'écarts-types à la vérité terrain.

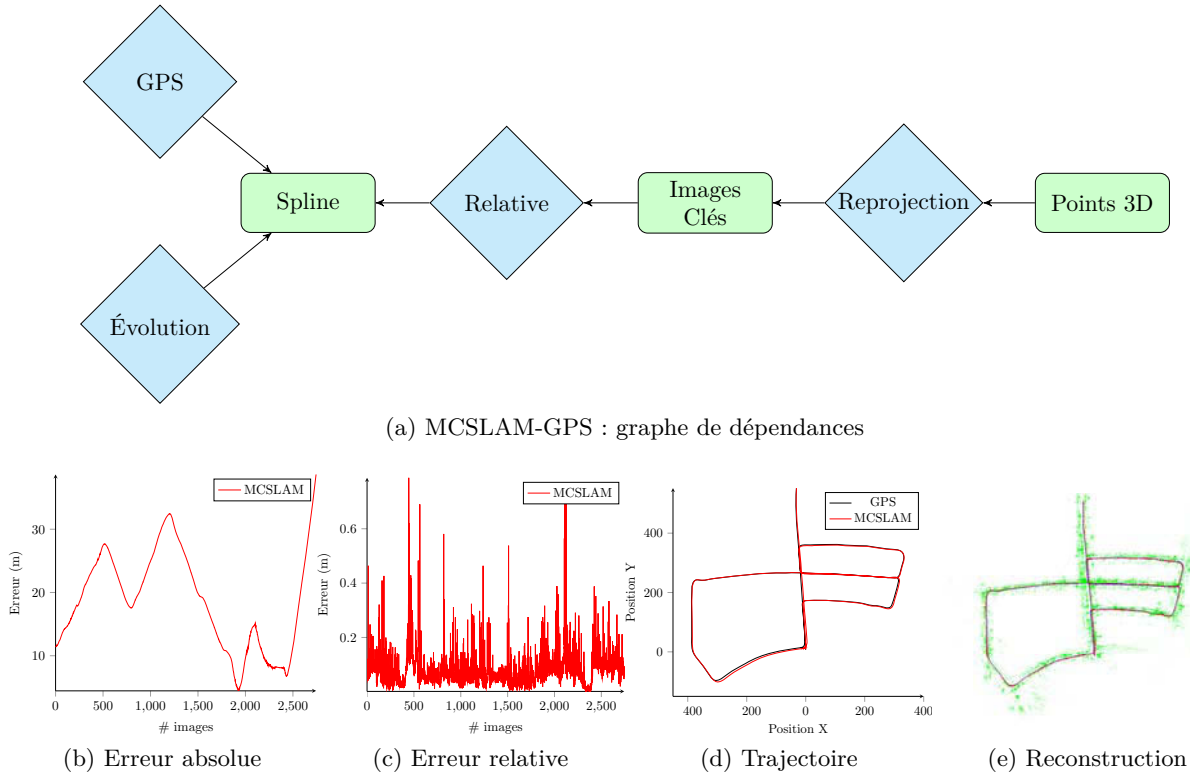
11.2.1 Contrainte au gps relatif

Cette première expérimentation illustre la contrainte en déplacement relatif entre la spline et les images clés de la caméra. La contrainte utilisée entre la spline et le GPS est une contrainte absolue, tandis que la contrainte entre la spline et les poses caméras minimise la différence de déplacement relatif vu dans la section 10.6. L'intérêt de cette approche est d'appliquer une contrainte entre deux trajectoires (spline et poses caméra) qui n'évoluent pas forcément dans le même repère. Ainsi, on veut montrer que cette contrainte relative, entre la spline et les images clés, est suffisante pour corriger la reconstruction 3D. L'algorithme de MCSLAM est généré avec le graphe de dépendances correspondant à la figure 11.4a. La figure 11.4 montre les résultats de cette approche sans que les poses clés ne soient contraintes à suivre la spline en position absolue. Celle-ci offre des résultats proches de la vérité terrain. De plus, le temps de traitement est de 10 images par seconde. Cette contrainte relative sera utilisée dans le reste des expérimentations entre la spline et les poses caméra.

11.2.2 Application à la navigation par caméra et centrale inertielle

Les expérimentions qui suivent illustrent la précision du MCSLAM contraint par une caméra et une centrale inertielle, noté MCSLAM-IMU, et correspondant au graphe de dépendances de la figure 11.5b. Il est comparé au SLAM monoculaire qui est simplement noté MCSLAM et qui correspond au graphe de dépendances de la figure 11.5a. Le MCSLAM-IMU utilise à la fois la vision et la centrale inertielle pour construire la spline. La contrainte entre la spline et les poses clés est la contrainte en déplacement relatif présentée dans la section 10.6. La contrainte inertielle appliquée à la spline est celle décrite dans la section 10.4. On inclut également un modèle d'évolution dont la pondération $\sigma_{evolution}$ est faible.

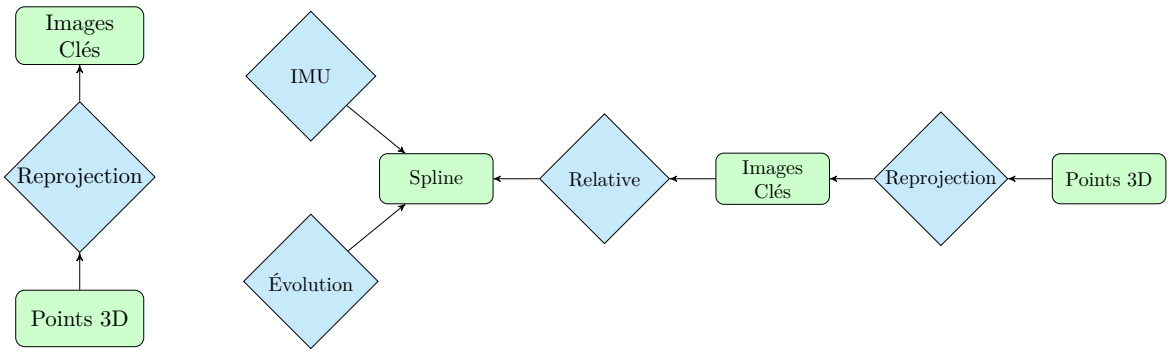
Les expérimentations sur les 4 séquences 11.6, 11.7, 11.8 et 11.9 montrent les résultats de reconstructions et les écarts à la vérité terrain des méthodes MCSLAM et MCSLAM-IMU. On observe sur les figures des 3 premières expérimentations 11.6, 11.7, 11.8 que la dérive du MCSLAM-IMU est significativement moins importante que celle du MCSLAM. La contrainte inertielle donne une information d'accélération métrique, donc limite la dérive de l'échelle. C'est pourquoi, la dérive absolue est nettement réduite. L'erreur relative met en avant la variation du facteur d'échelle qui est atténué avec le MCSLAM-IMU. Toutefois, le temps de traitement de l'ordre de 3 images par seconde n'est pas temps réel. Dans l'expérimentation 11.7, on observe un écart important sur les graphes d'erreurs absolues et relatives autour de la la 1000^e image. Cela est dû au GPS qui capte mal sur une centaine de mètres et qui fausse la vérité terrain. Dans ces conditions, une localisation par GPS aurait échoué, tandis que le



(a) MCSLAM-GPS : graphe de dépendances

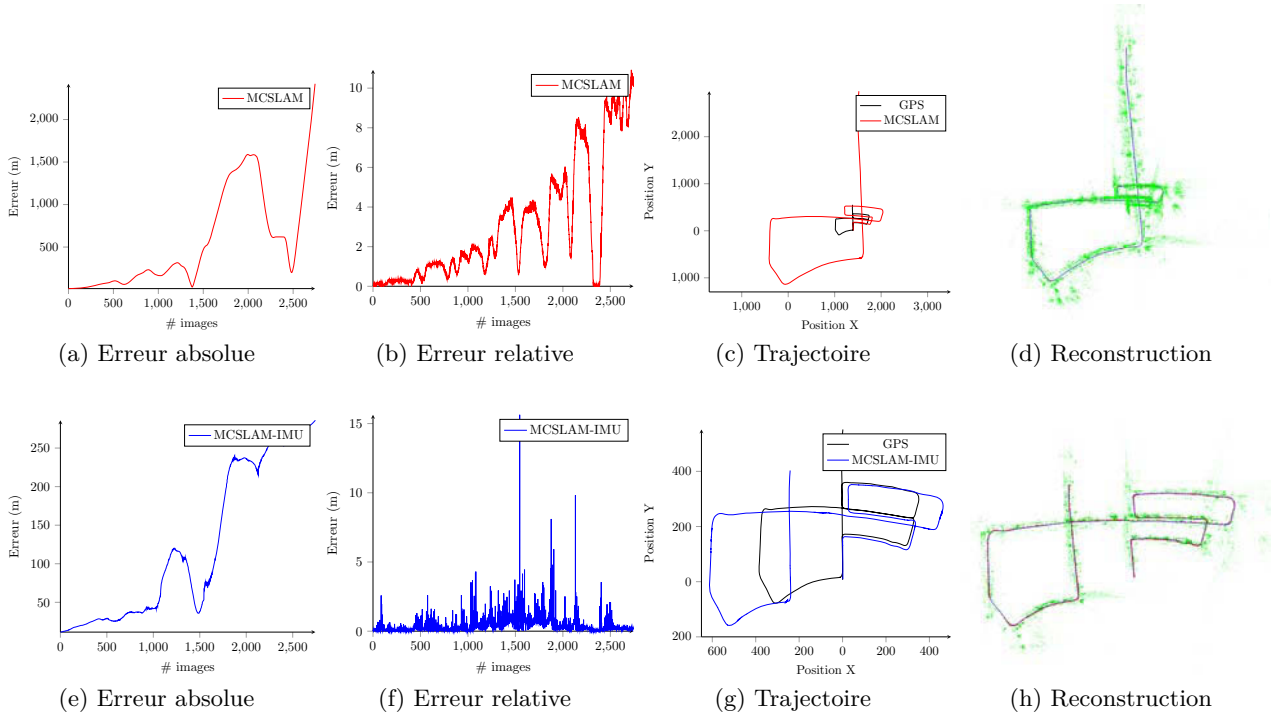
(b) Erreur absolue (c) Erreur relative (d) Trajectoire (e) Reconstruction

FIGURE 11.4 – Expérimentation avec contrainte relative entre la spline et les images clés sur le jeu de données 09_30_18.



(a) MCSLAM (b) MCSLAM-IMU

FIGURE 11.5 – Graphes de dépendances utilisés pour évaluer l'apport d'une centrale inertielle (MCSLAM-IMU) par rapport au SLAM classique (MCSLAM).



(a) Erreur absolue (b) Erreur relative (c) Trajectoire (d) Reconstruction

(e) Erreur absolue (f) Erreur relative (g) Trajectoire (h) Reconstruction

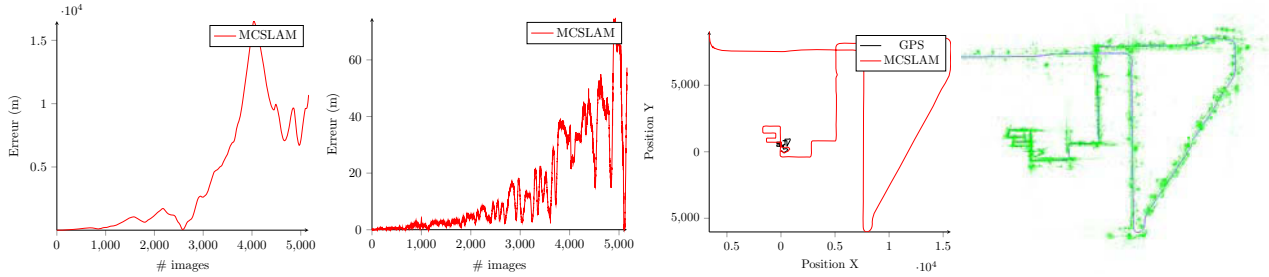
FIGURE 11.6 – Expérimentations sur le jeu de données 09_30_18

MCSLAM-IMU se comporte bien. La 4^{e} expérience 11.9 montre également de meilleurs résultats pour le MCSLAM-IMU mais dans une moindre mesure. Il est probable que cette différence provienne de la trajectoire. Le facteur d'échelle de reconstruction varie très peu tant que des amers correctement géo-référencés sont observés. Les variations les plus importantes du facteur d'échelle ont lieu dans les virages serrés. C'est la conséquence d'un faible champ recouvrant entre les images consécutives. Cela provoque la création de nouveaux amers 3D. C'est pourquoi, il est probable que la faible variation du facteur d'échelle sur la trajectoire 09_30_34 soit la conséquence du faible nombre de virages serrés.

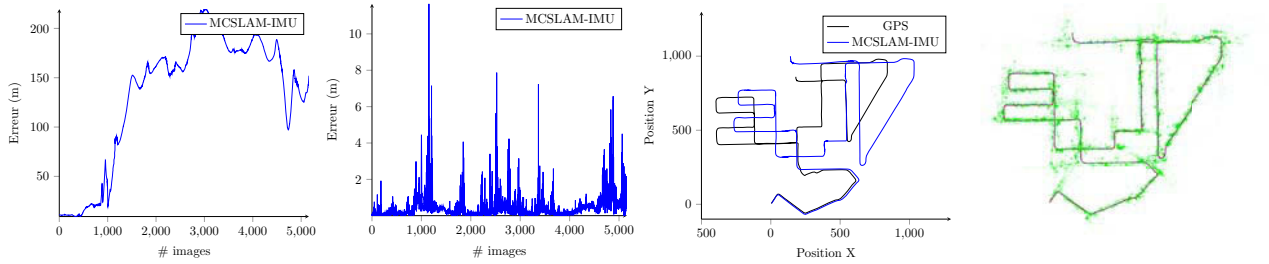
11.2.3 Application au calibrage hors-ligne

Les expérimentations présentées ici illustrent l'utilisation du MCSLAM pour effectuer du calibrage. Comme le montre les précédentes expérimentations, le SLAM monoculaire subit une dérive du facteur d'échelle. L'idée est d'utiliser le MCSLAM avec une contrainte sur le mouvement, ici le GPS, pour empêcher la dérive. Puis on relâche les paramètres intrinsèques de la caméra afin qu'ils soient optimisés. Les nouveaux paramètres intrinsèques obtenus sont évalués avec l'algorithme de SLAM monoculaire sans contrainte de mouvement.

On utilise le MCSLAM pour optimiser les paramètres intrinsèques, notés $\mathbf{K}$. Pour cela, on utilise le résultat de l'algorithme MCSLAM-IMU de l'expérimentation 11.6 sur la trajectoire 09_30_18 et on effectue une étape hors-ligne d'optimisation globale : la spline est contrainte par les données GPS et les poses caméra. Ici, on utilise la contrainte de pose absolue qui contraint les poses caméra à être précisément sur la spline. Le graphe de dépendances utilisé est illustré par la figure 11.10a. Les paramètres intrinsèques $\mathbf{K}$ de la caméra sont relâchés. Les données de calibrage fournies par la base KITTI pour la caméra 00 de la séquence 09_30_18 sont (au format du modèle unifié vue dans la

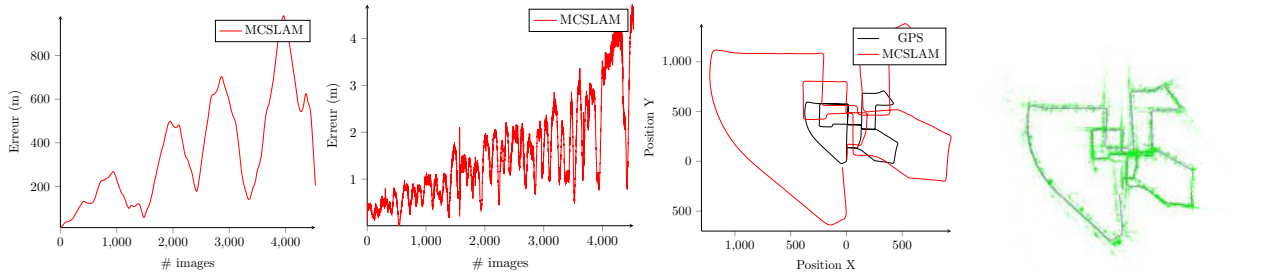


(a) Erreur absolue (b) Erreur relative (c) Trajectoire (d) Reconstruction

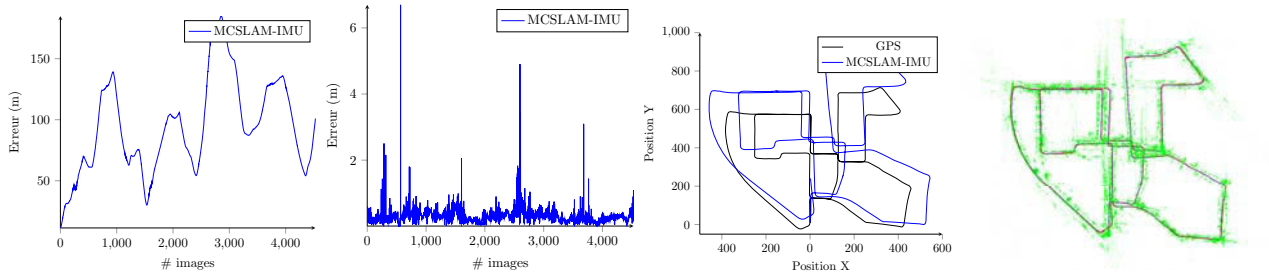


(e) Erreur absolue (f) Erreur relative (g) Trajectoire (h) Reconstruction

FIGURE 11.7 – Expérimentations sur le jeu de données 09_30_28

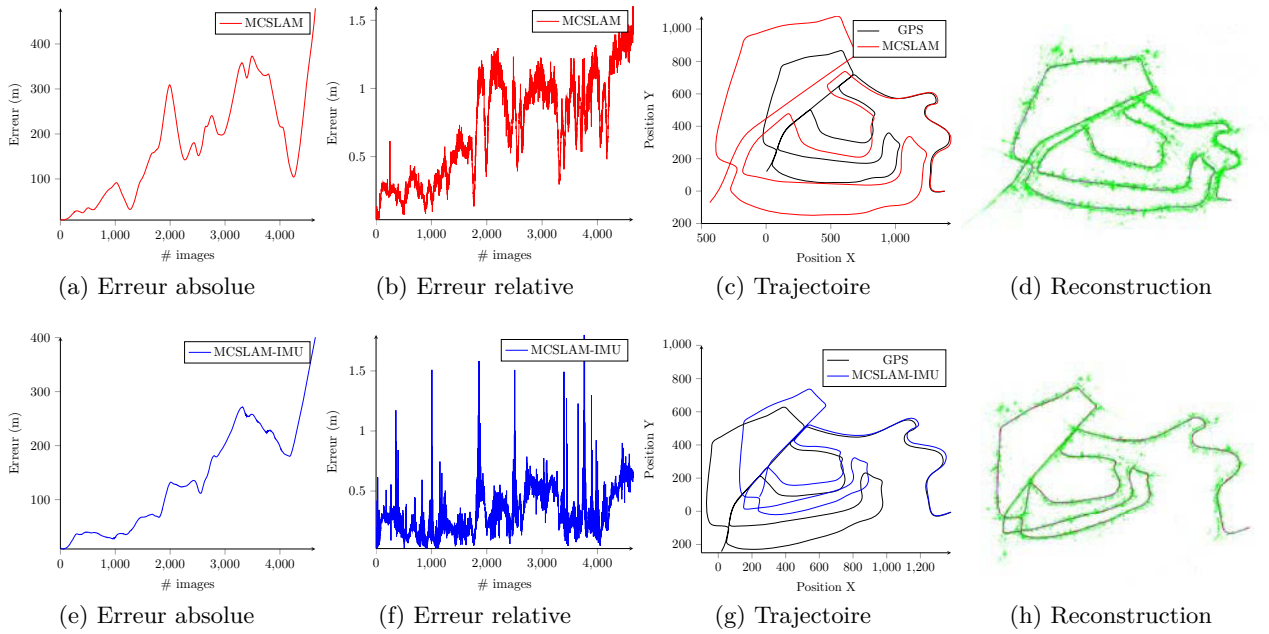


(a) Erreur absolue (b) Erreur relative (c) Trajectoire (d) Reconstruction



(e) Erreur absolue (f) Erreur relative (g) Trajectoire (h) Reconstruction

FIGURE 11.8 – Expérimentations sur le jeu de données 10_03_27



(a) Erreur absolue (b) Erreur relative (c) Trajectoire (d) Reconstruction

(e) Erreur absolue (f) Erreur relative (g) Trajectoire (h) Reconstruction

FIGURE 11.9 – Expérimentations sur le jeu de données 10_03_34

Séquence	09_30_18		09_30_28		10_03_34		10_03_27	
Erreur (m)	Absolue	Relative	Absolue	Relative	Absolue	Relative	Absolue	Relative
MCSLAM-GPS	18/6.5	0.07/0.04	∅	∅	∅	∅	∅	∅
MCSLAM	558/478	3.05/2.46	4081/3998	14/13	359/207	1.51/0.80	176/95	0.71/0.32
MCSLAM-IMU	123/91	0.39/0.33	132/54	0.38/0.30	94/31	0.30/0.12	137/81	0.30/0.14
MCSLAM-Calib	28/4.78	0.08/0.04	98/46	0.28/0.16	∅	∅	∅	∅

TABLE 11.2 – Récapitulatif des expérimentations : erreurs moyennes / écarts types à la trajectoire.

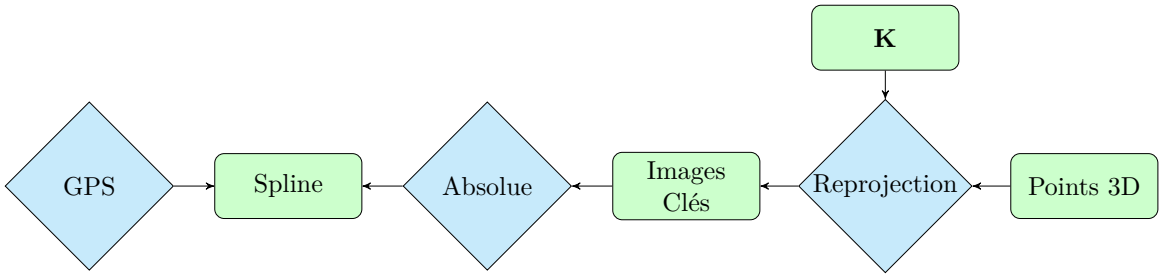
section 1.2.3) :

$$\mathbf{K} = \begin{bmatrix} 707.09 & 0 & 601.88 \\ 0 & 707.09 & 183.11 \\ 0 & 0 & 1 \end{bmatrix}, \quad \xi = 0 \tag{11.1}$$

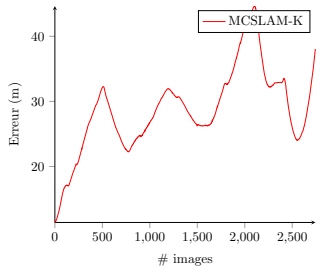
Après optimisation des paramètres intrinsèques sur le jeu de données 09_30_18, on obtient les valeurs suivantes :

$$\mathbf{K} = \begin{bmatrix} 675.88 & 0 & 605.0 \\ 0 & 675.88 & 185.6 \\ 0 & 0 & 1 \end{bmatrix}, \quad \xi = -0.04029 \tag{11.2}$$

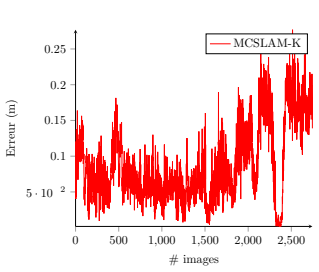
Puis, on relance les méthodes de SLAM monoculaire, en utilisant uniquement la contrainte d'erreur de reprojection, pour évaluer les nouveaux paramètres sur la séquence utilisée pour le calibrage 09_30_18 mais également sur la seconde séquence 09_30_28. La méthode, utilisant les nouveaux paramètres intrinsèques, est notée MCSLAM-K. On obtient les trajectoires correspondant aux figures 11.10d et 11.10h. On observe que les écarts à la vérité terrain sont moins importants que précédemment et la dérive est réduite, à la fois sur la séquence qui a servi à l'étalonnage que sur la seconde séquence.



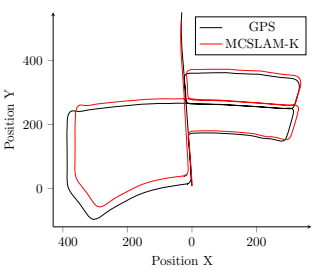
(a) Graphe de dépendances



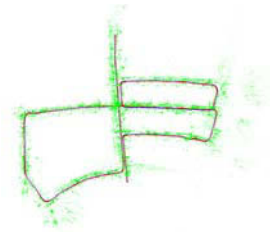
(b) Erreur absolue



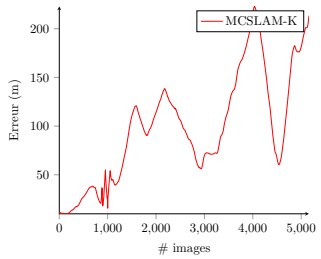
(c) Erreur relative



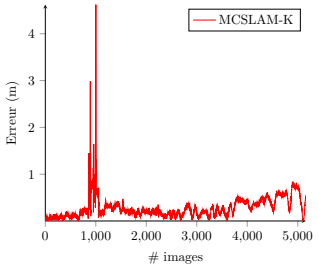
(d) Trajectoire



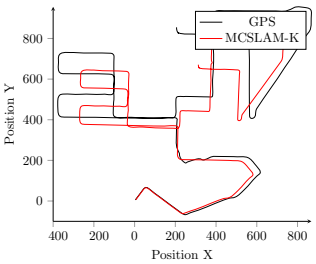
(e) Reconstruction



(f) Erreur absolue



(g) Erreur relative



(h) Trajectoire

(i) Reconstruction

FIGURE 11.10 – Expérimentations sur les jeux de données 09_30_18 et 09_30_28.

11.3 Conclusion

Toutes les variantes de MCSLAM utilisées dans ces expérimentations (contraintes de structure et de mouvement) ont été générées avec le MCSLAM. D'une expérimentation à l'autre, la définition de l'algorithme d'optimisation est réalisée en une seule ligne de code correspondant à la définition du graphe de dépendances. Par exemple, le graphe de dépendances de l'algorithme MCSLAM-IMU est défini de la façon suivante :

```
Graph<
     Constraints(Reprojection,IMU,Relative,Evolution),
     Parameters(Spline,KeyFrames,Points3D)
    > graph;
```

La modularité du *framework* permet de gérer différentes configurations de SLAM, avec des objets 3D ou des capteurs différents. Les premières expérimentations ont montré que des contraintes de structure permettent l'estimation précise d'objets 3D présents dans l'environnement pour des applications de Réalité Augmentée. Des expérimentations appliquées à la navigation multi-capteurs en environnements extérieurs ont montré la capacité du MCSLAM à utiliser conjointement différentes contraintes de mouvement, jusqu'à effectuer du calibrage de paramètres intrinsèques. Les données nécessaires à l'utilisation de la contrainte inertielle (les biais du gyromètre, de l'accéléromètre ainsi que le vecteur gravité) ont également été déterminées avec un processus de calibrage hors-ligne.

En considérant le temps réel à $10Hz$ et en utilisant l'intégralité des données inertielles (cadencées à $100Hz$) sur une fenêtre d'optimisation de 10 secondes, les performances d'exécution de l'algorithme MCSLAM-IMU ne sont pas temps réel. A titre d'exemple, l'optimisation incrémentale de 20 noeuds, 10 images clés, 1300 points 3D associés à 7400 erreurs de reprojection et 1000 données inertielles (en considérant également le modèle d'évolution, et la contrainte entre poses clés et spline), le temps nécessaire pour effectuer 3 itérations de Levenberg-Marquardt avec la bibliothèque LMA sur un seul *thread* est de 0.3 seconde. La majeure partie de ce temps (0.28 seconde) est passée dans les calculs de dérivées numériques (seules les erreurs de reprojection ont été implémentées avec des dérivées analytiques). Une solution est de paralléliser le calcul des dérivées. Toutefois, cela entraînerait une sur-consommation de CPU alors qu'il serait plus judicieux d'implémenter les dérivées analytiques.

L'utilisation de la contrainte inertielle (sans le GPS) s'est révélée difficile. En effet, l'accéléromètre doit fixer le facteur d'échelle de la spline, ce que ne fait pas la caméra. Dans le but de corriger la dérive de la reconstruction, on attribue une pondération plus importante à la centrale inertielle. Toutefois, les données inertielles seules ne sont pas suffisantes pour construire précisément la spline. Ainsi, le réglage des pondérations entre contraintes reste problématique dès lors que l'on change de capteur ou que l'on ajoute une nouvelle contrainte.

Conclusion

Ce chapitre a présenté le MCSLAM qui est un *framework* modulaire pour le développement d'algorithmes de SLAM. Les expérimentations ont montré l'utilisation du *framework* avec différentes contraintes et 3 types de capteurs (caméra, centrale inertielle et GPS). Pour des applications de Réalité Augmentée, des contraintes de structure relativement simples estiment efficacement la pose de plusieurs objets en temps réel. Sur les jeux de données de la base KITTI, les contraintes de mouvement limitent (dans le cas de la centrale inertielle) ou corrigent complètement (avec le GPS) la dérive de la trajectoire et donc de la reconstruction 3D. Du plus, la configuration du MCSLAM s'effectue simplement par modification du graphe de dépendances. Le développement d'applications de SLAM est facilité par les méthodes de génération automatique de code utilisées.

Une représentation continue de la trajectoire est présentée pour fusionner les différentes données de positions asynchrones. L'approche se révèle précise et temps réel pour fusionner des données GPS et la trajectoire caméra. La dérive de la reconstruction 3D est alors corrigée. Toutefois, l'approche proposée avec une centrale inertielle en complément de la caméra donne des résultats mitigés. Certes, la dérive du facteur d'échelle est réduite, mais compte tenu de la bonne qualité des données inertielles, de meilleurs résultats étaient attendus. La spline se révèle être un outil précis pour connaître la position du système dans le passé. Mais les applications visées nécessitent une localisation précise et temps réel. Les temps de calculs sont également à améliorer, notamment sur les calculs de dérivées liés aux contraintes inertielles. Ces points nécessitent des travaux supplémentaires.

Conclusion et perspectives

Conclusion

La problématique abordée dans cette thèse est l'amélioration de la localisation d'un système de perception composé au moins d'une caméra. La solution choisie est d'ajouter des contraintes dans un algorithme de SLAM visuel par ajustement de faisceaux. L'estimation des paramètres est réalisée avec l'algorithme de Levenberg-Marquardt. Chaque contrainte ajoute une information supplémentaire sur l'estimation des paramètres donc sur les dérivées. Cela modifie la structure interne des équations normales. L'écriture efficace, avec une implémentation spécialisée, de tels algorithmes devient impossible dès lors que le nombre de contraintes est important.

C'est pourquoi, le premier travail de la thèse a été de concevoir la bibliothèque LMA dont le rôle est d'ajouter un niveau d'abstraction, sans perte de performance, à l'implémentation de l'algorithme de Levenberg-Marquardt. Cette bibliothèque agit comme un programme qui génère un solveur spécialisé, donc optimisé, quel que soit le degré de difficulté dû au nombre de contraintes et aux types de paramètres. Des comparatifs exhaustifs ont montré des performances supérieures aux alternatives de l'état de l'art proposant des fonctionnalités semblables. De plus, la bibliothèque est mise à disposition de la communauté en *open-source* : github.com/bezout/LMA.

LMA se révèle performant. Mais cela se fait au prix de la modularité : en effet, la fonction de coût à minimiser doit être connue au moment de la compilation. C'est pourquoi, la conception du *framework* MCSLAM est basée sur l'utilisation d'un graphe de dépendances, connu à la compilation, qui lie les paramètres aux contraintes. Le MCSLAM analyse automatiquement ce graphe pour générer le solveur adéquat avec LMA. La création d'une application de SLAM à contraintes multiples se fait alors par ajout de contraintes dans le graphe. Cette approche offre à la fois une vue d'ensemble sur la fonction de coût et permet de la modifier facilement (par ajout et suppression des contraintes dans le graphe). Le MCSLAM a été utilisé pour générer plusieurs types d'applications utilisant des contraintes de structure ou de mouvement.

Pour des applications de Réalité Augmentée, on utilise des contraintes géométriques correspondant à des modèles d'objets 3D approximativement connus et initialisés en ligne. Les expérimentations ont montré que l'utilisation de contraintes simples, entre les objets et les amers 3D, améliore non seulement la localisation mais également l'estimation de la pose (et des dimensions) des objets. L'approche offre des performances d'exécution de l'ordre de quelques millisecondes par image sur un ordinateur de bureau, ce qui laisse supposer des performances également temps réel sur plateforme mobile. La précision du recalage entre les objets virtuels et réels offre un rendu visuel de qualité pour des applications de Réalité Augmentée.

Dans le cadre d'application de navigation autonome, des capteurs tels qu'un GPS et une centrale inertielle apportent des contraintes sur le mouvement du système. Ces données étant à la fois asynchrones et/ou de natures différentes (vitesse angulaire, accélération), on utilise une représentation continue de la trajectoire modélisée par une spline. Ainsi, l'intégralité des capteurs contribue à l'estimation d'une même trajectoire. De plus, la contrainte de mouvement donnée par la spline est répercutée sur les images clés du SLAM visuel. Avec le GPS, le MCSLAM montre une bonne précision. En utilisant uniquement la caméra et la centrale inertielle, la dérive de la reconstruction est limitée.

Perspectives

Au cours de cette thèse, les fonctionnalités et les performances de la bibliothèque LMA ont toujours été à la hauteur des problèmes posés. Toutefois, des améliorations sont à prévoir. La structure de données des équations normales est purement éparse. Une implémentation pour les problèmes denses est à prévoir. De plus, la minimisation par région de confiance améliorera la vitesse de convergence. Il est également nécessaire d'implémenter l'algorithme de Cholesky éparse par bloc pour une résolution plus rapide sur des problèmes éparses de petites et moyennes dimensions. La ré-écriture d'une partie du code de LMA, en utilisant des bibliothèques récentes [9] de méta-programmation, apporterait une meilleure lisibilité du code et un temps de compilation plus rapide. LMA n'utilise pas explicitement les capacités des architectures parallèles. Des outils modernes comme NT2 [10] permettent d'exploiter de telles architectures tout en conservant un haut niveau d'expressivité du langage. Ainsi, l'utilisation de NT2 pour paralléliser implicitement les opérations matricielles avec le SIMD ou le GPU est une perspective prometteuse.

Dans le cadre de la Réalité Augmentée, de nombreuses améliorations sont envisageables. La plus importante est l'initialisation automatique des objets afin de supprimer l'étape de sélection manuelle dans l'image. Avec l'ajout des objets se pose également la question de leur suppression. Est-il possible de déterminer qu'un objet est devenu inutile ? Pour améliorer l'interactivité avec la scène, le recalage ou « suivi » d'objets en mouvement est également une problématique intéressante. Cela nécessite de considérer que l'environnement 3D est composé d'objets rigides mais dynamiques. Le problème de la détection des occultations, afin d'adapter dynamiquement le rendu est également un problème soulevé, qui est généralement résolu avec des capteurs de profondeur. Dans ce travail, les contraintes sur les objets sont très simples. Elles consistent à minimiser la distance entre les amers 3D et les objets. Des approches plus élaborées sont possibles, notamment en supposant les points à la surface des objets ce qui permet de supprimer des degrés de liberté. Les contraintes entre segments et objets courbes constituent également une problématique rencontrée dans ce travail. L'ajout d'un capteur de profondeur, dans le MCSLAM, afin de contraindre la profondeur des amers 3D constitue également une perspective intéressante.

L'utilisation d'une représentation continue de la trajectoire pour fusionner des données capteurs asynchrones a été la solution choisie dans ce travail. En pratique, la spline permet de fusionner des données capteurs et de contraindre la reconstruction 3D. Toutefois, des améliorations sont nécessaires. L'utilisation d'une méthode de pondération automatique des différentes contraintes est indispensable. De plus, la création des nœuds soulève la question de l'utilisation en temps réel ou en temps différée de la spline. La seconde solution a pour conséquence une bonne estimation de la trajectoire passée mais limite l'intérêt de la spline pour estimer la pose courante. Les temps de traitement importants des capteurs rapides tels que les centrales inertielles imposent l'utilisation de dérivées analytiques. Plus d'expérimentations sont nécessaires concernant les contraintes de mouvement afin de déterminer si l'approche est viable dans des contextes multi-capteurs où le temps réel est requis.

Dans toutes les expérimentations effectuées, les capteurs étaient de bonnes qualités. Une évaluation des approches en utilisant des capteurs bas coûts, comme les caméras ou les centrales inertielles de téléphone portable, est nécessaire pour être exploitables en contexte industriel. Le dernier point proposé en perspective est la réalisation d'une application regroupant à la fois des contraintes de mouvement et des contraintes de structure.

9. github.com/boostorg/hana, github.com/edouarda/brigand : bibliothèques de méta-programmation destinées à remplacer Boost.MPL et Boost.Fusion.
10. github.com/jfalcou/nt2 : bibliothèque matricielle avec la syntaxe Matlab.

Annexe

11.4 Types matriciels : allocation statique et dynamique

Cette section présente une expérimentation afin d'illustrer les différences de performances en temps pour effectuer des calculs avec des matrices allouées dynamiquement (avec un appel à l'opérateur **new** du C++) ou statiquement (sur la pile d'appels du programme). L'objectif est de montrer que lorsque les dimensions des matrices sont connues (et qu'elles sont de petites tailles), il est avantageux d'utiliser cette information à la compilation. Cette expérimentation s'effectue sur un produit matriciel qui est une opération appelée un très grand nombre de fois dans les problèmes d'ajustement de faisceaux. Les bibliothèques matricielles testées sont Eigen, Toon et Blaze. Les résultats sont dans la table 11.3.

On remarque que quelle que soit la bibliothèque matricielle utilisée, le temps d'exécution est 3 à 10 fois plus rapide en utilisant les matrices statiques. Toutefois, cette différence tend à diminuer voir s'inverser lorsque la dimension des matrices augmente. De plus, la quantité d'octets pouvant être allouée directement sur la pile d'exécution est limitée par la capacité mémoire de cette pile.

11.5 Matrice éparse

On a vu dans la section précédente que les hessiennes sont fortement éparses, il est donc nécessaire d'utiliser une structure de données adaptée. Le principe du codage éparse de matrice est de ne considérer que les valeurs non nulles pour réduire la quantité de mémoire requise et également éviter les calculs inutiles avec les valeurs nulles. On présente dans cette section différentes approches couramment utilisées pour le codage de matrice éparse. Chaque représentation sera illustrée sur la

$M_{9\times3} = M_{9\times3} \times M_{3\times3}$	Blaze	TooN	Eigen
Dynamique	0.053281	0.029788	0.035122
Statique	0.018956	0.009159	0.016205
$M_{6\times3} = M_{6\times3} \times M_{3\times3}$	Blaze	TooN	Eigen
Dynamique	0.043416	0.024538	0.032107
Statique	0.012977	0.006447	0.006427

TABLE 11.3 – Temps de calcul pour effectuer un produit matriciel et affecter le résultat pour différentes bibliothèques en fonction d'une allocation de la mémoire dynamique ou statique. On remarque que quel que soit la bibliothèque utilisée, les calculs faisant intervenir des matrices statiques sont toujours plus rapides d'un facteur 4 à 5.

matrice A :

$$A = \begin{bmatrix} 1 & 2 & 3 & 0 & 0 & 0 & 0 \\ 4 & 5 & 6 & 0 & 0 & 0 & 0 \\ 7 & 8 & 9 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 10 & 11 & 0 & 0 \\ 0 & 0 & 0 & 12 & 13 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 14 & 15 \\ 0 & 0 & 0 & 0 & 0 & 16 & 17 \end{bmatrix} \tag{11.3}$$

11.5.1 Compression par ligne

La compression par ligne (en anglais *CRS* pour *Compressed Row Storage*) est la représentation la plus utilisée notamment dans les bibliothèques de calcul matriciel éparse (CSparse [11], Eigen [12]). Cela consiste à représenter une matrice M par deux ensembles, un ensemble de valeurs et un ensemble d'indices. L'ensemble des valeurs correspond à chacune des lignes de la matrice M à laquelle on a retiré tous les zéros. L'ensemble des indices conserve la position de chacune des valeurs dans les lignes de la matrice M.

$$CRS(M) = \begin{Bmatrix} Valeurs & Indices \end{Bmatrix} \tag{11.4}$$

Appliquée à la matrice A, on obtient la représentation suivante :

$$CRS(A) = \left\{ \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & \\ 12 & 13 & \\ 14 & 15 & \\ 16 & 17 & \end{pmatrix} \quad \begin{pmatrix} 0 & 1 & 2 \\ 0 & 1 & 2 \\ 0 & 1 & 2 \\ 3 & 4 & \\ 3 & 4 & \\ 5 & 6 & \\ 5 & 6 & \end{pmatrix} \right\} \tag{11.5}$$

On observe que ce stockage génère autant d'indices que de valeurs non nulles dans la matrice.

11.5.2 Compression par bloc et par ligne

Un bloc correspond à une sous matrice ne contenant pas de valeur nulle. Ainsi, la compression par bloc et par ligne (notée *BCSR*) consiste à regrouper, lorsque cela est possible, les valeurs non nulles contiguës dans des mêmes blocs et d'associer à chaque bloc une position et une dimension :

$$BCRS(M) = \begin{Bmatrix} Blocs & Indices & Dimension \end{Bmatrix} \tag{11.6}$$

Cette compression se réalise en deux étapes : tout d'abord, on regroupe les valeurs nulles et non nulles dans des blocs différents de dimensions variables, puis on supprime les blocs nulles.

$$A = \begin{bmatrix} \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} & \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \\ \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} & \begin{bmatrix} 10 & 11 \\ 12 & 13 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} & \begin{bmatrix} 14 & 15 \\ 16 & 17 \end{bmatrix} \end{bmatrix} \tag{11.7}$$

11. sparsematrixe
12. tux.eigen.org

Pour chacun des blocs obtenus, on conserve la position du premier élément du bloc dans la matrice A ainsi que la dimension :

$$BCRS(A) = \left\{ \begin{pmatrix} \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \\ \begin{bmatrix} 10 & 11 \\ 12 & 13 \end{bmatrix} \\ \begin{bmatrix} 14 & 15 \\ 16 & 17 \end{bmatrix} \end{pmatrix} \quad \begin{pmatrix} (0,0) \\ (3,3) \\ (5,5) \end{pmatrix} \quad \begin{pmatrix} (3 \times 3) \\ (2 \times 2) \\ (2 \times 2) \end{pmatrix} \right\} \tag{11.8}$$

Cette compression permet de réduire de manière importante l'ensemble des indices. Toutefois, il est important de remarquer que l'utilisation de cette représentation pour effectuer des calculs matriciels n'est efficace qu'avec des matrices à structure compatible (*i.e.* structure identique pour l'addition et structure transposée pour le produit matriciel).

11.5.3 Compression par ensemble de blocs de même dimension et par ligne

Cette dernière représentation consiste à regrouper ensemble les blocs de dimension identique. Ainsi, cette représentation (notée *SBCSR* pour *Static Size Block CSR*) se compose d'autant de listes de blocs que la matrice M contient de blocs de dimensions différentes, et comme précédemment, on conserve la position dans M de chacun des blocs :

$$SBCRS(M) = \left\{ \begin{array}{ccc} BCRS_{h_0 \times l_0} = & Blocs_{h_0 \times l_0} & Indices_0 \\ & \vdots & \\ BCRS_{h_n \times l_n} = & Blocs_{h_n \times l_n} & Indices_n \end{array} \right\} \tag{11.9}$$

Appliquée à la matrice A, cette représentation se compose d'une liste de blocs de dimension 3×3 et d'une liste de blocs de dimension 2×2 :

$$SBCRS(A) = \left\{ \begin{array}{ll} BCRS_{3\times3} = \begin{pmatrix} \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \end{pmatrix} & \Big((0,0) \Big) \\ BCRS_{2\times2} = \begin{pmatrix} \begin{bmatrix} 10 & 11 \\ 12 & 13 \end{bmatrix} \\ \begin{bmatrix} 14 & 15 \\ 16 & 17 \end{bmatrix} \end{pmatrix} & \begin{pmatrix} (3,3) \\ (5,5) \end{pmatrix} \end{array} \right\} \tag{11.10}$$

Cette représentation permet entre autres d'économiser le stockage redondant de la dimension de chaque bloc mais surtout d'optimiser le code généré si la dimension des blocs est connue à la compilation. Lorsqu'il s'agit de petites matrices, l'utilisation de l'allocation statique est plus performante que l'allocation dynamique comme le montre l'expérimentation 11.4. De plus, connaître la dimension des matrices à la compilation permet à la bibliothèque matricielle de spécifier certains calculs et au compilateur d'utiliser des optimisations spécialisées pour la dimension donnée. Toutefois, il n'est pas possible de stocker des blocs de matrices de tailles différentes dans un même conteneur sans utiliser d'abstraction (donc sans dégrader les performances). C'est pourquoi, il est nécessaire de regrouper dans des mêmes conteneurs les blocs de même taille.

11.6 Comparatifs entre g2o, Ceres et LMA

Le tableau 11.4 liste des caractéristiques des jeux de données utilisés. Les tableaux 11.7, 11.5, 11.6 et 11.8 montrent les différences de temps de calcul et de précision entre les solveurs g2o, Ceres et LMA

	#Paramètres	#résidus	#ddl_1	#ddl_2	#$blocs_1$	#$blocs_2$
Cercle	3	100000	3	0	1	0
Sphere	594409	198135	4	3	1	198135
Bal_0	23769	63686	9	3	49	7776
Bal_1	34134	72910	9	3	21	11315
Bal_2	33753	92244	9	3	73	11032
Bal_3	66462	167436	9	3	16	22106
Bal_4	169665	408404	9	3	215	55910
Bal_5	197709	451822	9	3	257	65132
Bal_6	184446	574902	9	3	93	61203
Bal_7	192627	694346	9	3	52	64053
Bal_8	193686	767874	9	3	88	64298
Bal_9	350541	1020422	9	3	1064	113655
Bal_{10}	304650	1068816	9	3	394	100368
Bal_{11}	288813	1136238	9	3	150	95821
Bal_{12}	409173	1175884	9	3	1266	132593
Bal_{13}	423297	1225186	9	3	1340	137079
Bal_{14}	448818	1283292	9	3	1469	145199
Bal_{15}	455577	1302960	9	3	1514	147317
Bal_{16}	466818	1326578	9	3	1587	150845
Bal_{17}	476238	1342536	9	3	1642	153820
Bal_{18}	485013	1357436	9	3	1723	156502
Bal_{19}	400206	1503304	9	3	202	132796
Bal_{20}	598422	2182772	9	3	245	198739
Bal_{21}	683394	2510536	9	3	356	226730
Bal_{22}	960174	3307624	9	3	3068	310854
Bal_{23}	569958	3385950	9	3	961	187103
Bal_{24}	934995	3398290	9	3	427	310384
Bal_{25}	1590279	5571954	9	3	871	527480
Bal_{26}	1637382	6117726	9	3	744	543562
Bal_{27}	2351304	8104680	9	3	1102	780462
Bal_{28}	2696298	9034252	9	3	1350	894716
Bal_{29}	2832342	9478062	9	3	1521	939551
Bal_{30}	2969322	9941884	9	3	1695	984689
Bal_{31}	2971941	9949262	9	3	1706	985529
Bal_{32}	2997771	10003892	9	3	1778	993923
Bal_{33}	1966443	10427466	9	3	1936	649673
Bal_{34}	4015011	18250250	9	3	4585	1324582

TABLE 11.4 – Caractéristiques des jeux de données utilisés dans les expérimentations :
paramètres : nombre de degrés de liberté
résidus : total d'erreurs (la dimension du vecteur de résidus)
ddl_1 : degrés de liberté pour une instance du premier type de paramètre
ddl_2 : degrés de liberté pour une instance du second type de paramètre
$blocs_1$: instances du premier type de paramètre
$blocs_2$: instances du second type de paramètre

	Temps			RMS		
Solveur	g2o	Ceres	LMA	g2o	Ceres	LMA
Sphere	3.398	8.048	**1.209**	0.000	0.000	0.000
Bal_0	3.570	1.978	**0.777**	0.653	**0.647**	0.650
Bal_1	3.653	1.500	**0.731**	0.912	**0.912**	0.912
Bal_2	5.739	2.826	**1.330**	0.608	0.608	**0.608**
Bal_3	8.469	3.132	**1.649**	0.464	**0.464**	0.464
Bal_4	33.805	16.240	**8.338**	0.672	**0.669**	0.672
Bal_5	41.139	21.372	**11.135**	0.660	**0.654**	0.660
Bal_6	35.717	15.451	**9.167**	0.711	**0.711**	0.711
Bal_7	39.287	21.175	**11.256**	0.834	0.838	**0.834**
Bal_8	45.339	24.749	**13.280**	0.870	**0.863**	0.866
Bal_9	220.238	82.155	**50.183**	0.751	**0.747**	0.751
Bal_{10}	73.998	43.498	**22.246**	0.665	**0.658**	0.664
Bal_{11}	103.711	54.632	**30.903**	0.662	**0.661**	0.662
Bal_{12}	162.777	81.714	**41.377**	0.945	**0.914**	0.943
Bal_{13}	230.421	115.814	**62.098**	0.627	**0.627**	0.627
Bal_{14}	361.391	171.062	**85.694**	**0.791**	0.794	0.794

TABLE 11.5 – Temps d'exécution total en secondes et RMS final de g2o, Ceres et LMA avec la méthode DENSE_SCHUR.

sur différents problèmes d'optimisation.

11.7 Paramétrisation d'un plan

La paramétrisation d'un plan pour l'optimisation n'est pas forcément un problème trivial. En effet, si l'on représente le plan par les paramètres de son équation, c'est-à-dire $ax + by + cz + d = 0$, alors il est sur-paramétré car seulement trois variables sont nécessaires. De plus, cette paramétrisation est dégénérée car toute multiplication par une constante définit le même plan. Une autre approche consiste à utiliser le vecteur normal $\overrightarrow{n}$ unitaire pour définir son orientation. Donc $\overrightarrow{n}$ est un élément de la sphère 3D unitaire aussi nommée S_2. Lors de l'optimisation, l'incrément définit un déplacement sur cette sphère relativement à $\overrightarrow{n}$. Or, la paramétrisation de cet incrément nécessite de définir un système de coordonnées dans le plan tangent à la sphère en $\overrightarrow{n}$. Cependant, ce système de coordonnées peut être construit à une rotation près autour de $\overrightarrow{n}$, ceci implique d'en choisir un aléatoirement et de le conserver. Finalement, la paramétrisation d'un vecteur unitaire pour l'optimisation nécessite un élément de S_2 et un élément de S_1 choisis arbitrairement et définissant le repère de coordonnées dans lequel s'exprime l'incrément. Or $S_2 \times S_1$ est homéomorphe à $SO(3)$, le groupe des matrices de rotation 3D qui est un groupe de Lie. Afin de le représenter, nous utiliserons la représentation par carte exponentielle locale déjà utilisée dans [20] et présentée dans la section 1.4 tout en fixant l'un des paramètres de la matrice anti-symétrique de manière arbitraire. De cette façon nous obtenons une paramétrisation non-dégénérée, minimale et *isotrope* des vecteurs unitaires 3D.

	Temps			RMS		
Solveur	g2o	Ceres	LMA	g2o	Ceres	LMA
Bal_0	3.331	1.284	**0.835**	**0.647**	0.649	0.647
Bal_1	3.622	1.670	**0.686**	0.912	**0.912**	0.912
Bal_2	4.963	2.256	**1.174**	0.608	0.611	**0.608**
Bal_3	8.455	3.587	**1.733**	0.464	**0.464**	0.464
Bal_4	22.705	9.149	**5.513**	0.670	0.682	**0.670**
Bal_5	25.091	10.103	**4.482**	**0.663**	0.665	0.693
Bal_6	33.773	14.345	**9.295**	0.711	0.711	**0.711**
Bal_7	39.108	20.459	**10.909**	**0.860**	0.866	0.874
Bal_8	44.047	26.165	**13.243**	0.866	**0.863**	0.864
Bal_9	89.232	55.407	**29.368**	0.759	**0.750**	0.758
Bal_{10}	68.062	31.143	**20.870**	0.665	**0.660**	0.664
Bal_{11}	89.971	40.910	**27.202**	0.662	0.664	**0.662**
Bal_{12}	133.826	72.183	**37.574**	0.947	**0.916**	0.947
Bal_{13}	151.084	71.805	**44.669**	0.631	0.645	**0.631**
Bal_{14}	213.635	108.831	**54.908**	0.809	**0.799**	0.799

TABLE 11.6 – Temps d'exécution total en secondes et RMS final de g2o, Ceres et LMA avec la méthode SPARSE_SCHUR.

	Temps		RMS	
Solveur	Ceres	LMA	Ceres	LMA
Sphere	6.711	**1.688**	4.497	**0.000**
Bal_0	2.642	**0.984**	**0.647**	0.647
Bal_1	3.047	**1.159**	0.912	**0.912**
Bal_2	3.874	**1.438**	0.609	**0.609**
Bal_3	6.456	**2.659**	0.464	**0.464**
Bal_4	18.065	**6.761**	**0.678**	0.678
Bal_5	18.034	**7.114**	**0.663**	0.666
Bal_6	24.702	**9.544**	**0.711**	0.711
Bal_7	21.634	**8.529**	**0.840**	1.008
Bal_8	25.424	**12.080**	**0.863**	0.866
Bal_9	23.427	**4.206**	**0.893**	1.769
Bal_{10}	41.924	**19.336**	0.750	**0.747**
Bal_{11}	41.404	**19.019**	**0.662**	0.665
Bal_{12}	29.478	**6.057**	**0.856**	2.882
Bal_{13}	24.309	**7.094**	**1.256**	1.884
Bal_{14}	34.240	**6.691**	**0.783**	3.792
Bal_{15}	29.889	**8.331**	**0.840**	1.556
Bal_{16}	36.440	**7.312**	**0.815**	3.299
Bal_{17}	31.220	**7.418**	**0.893**	3.152
Bal_{18}	26.087	**9.903**	3.973	**2.353**
Bal_{19}	52.356	**25.253**	**0.662**	0.662
Bal_{20}	56.077	**32.595**	**0.920**	0.956
Bal_{21}	96.875	**39.566**	**0.637**	0.638
Bal_{22}	72.935	**48.713**	1.204	**1.053**
Bal_{23}	121.805	**54.539**	**0.968**	0.971
Bal_{24}	132.540	**50.417**	**0.794**	0.798
Bal_{25}	206.993	**69.043**	**0.788**	0.791
Bal_{26}	245.047	**101.142**	**0.711**	0.713
Bal_{27}	281.227	**109.644**	**0.630**	0.698
Bal_{28}	277.463	**120.177**	**0.578**	0.582
Bal_{29}	352.847	**142.007**	**0.572**	0.573
Bal_{30}	362.979	**141.708**	**0.585**	0.592
Bal_{31}	347.778	**155.579**	**0.585**	0.596
Bal_{32}	309.591	**143.820**	**0.580**	0.580
Bal_{33}	421.227	**215.397**	**0.944**	0.944
Bal_{34}	639.210	**310.633**	**0.753**	0.806

TABLE 11.7 – Temps d'exécution total en secondes et RMS final de g2o, Ceres et LMA avec la méthode SPARSE.

	Temps		RMS	
Solveur	Ceres	LMA	Ceres	LMA
Bal_0	2.237	**0.978**	**0.647**	0.648
Bal_1	2.263	**1.111**	**0.912**	0.912
Bal_2	3.450	**1.439**	0.609	**0.608**
Bal_3	4.914	**2.484**	**0.464**	0.464
Bal_4	14.920	**6.587**	**0.673**	0.677
Bal_5	17.610	**6.402**	**0.658**	0.682
Bal_6	20.906	**9.502**	**0.711**	0.711
Bal_7	18.417	**10.804**	0.866	**0.850**
Bal_8	23.233	**12.210**	**0.863**	0.875
Bal_9	29.357	**7.413**	**0.819**	1.443
Bal_{10}	36.219	**17.804**	**0.747**	0.747
Bal_{11}	39.773	**19.369**	**0.658**	0.665
Bal_{12}	33.038	**11.998**	**0.798**	5.254
Bal_{13}	31.477	**9.105**	**0.854**	5.306
Bal_{14}	35.590	**10.003**	**0.789**	4.137
Bal_{15}	34.696	**14.845**	**0.819**	2.028
Bal_{16}	30.314	**15.599**	**1.443**	3.853
Bal_{17}	31.434	**13.974**	**2.545**	3.524
Bal_{18}	28.979	**15.422**	**1.199**	5.664
Bal_{19}	52.721	**24.854**	**0.661**	0.662
Bal_{20}	61.623	**32.160**	**0.916**	0.944
Bal_{21}	92.371	**42.278**	**0.631**	0.635
Bal_{22}	126.515	**55.620**	**1.002**	1.163
Bal_{23}	136.519	**58.821**	**0.968**	0.996
Bal_{24}	116.754	**53.399**	0.794	**0.791**
Bal_{25}	223.253	**92.821**	**0.786**	0.894
Bal_{26}	226.806	**106.393**	**0.711**	0.718
Bal_{27}	241.742	**120.091**	**0.644**	0.748
Bal_{28}	268.568	**120.425**	**0.581**	0.665
Bal_{29}	234.823	**158.733**	1.355	**0.577**
Bal_{30}	365.742	**176.849**	**0.580**	0.589
Bal_{31}	279.290	**161.615**	0.609	**0.599**
Bal_{32}	347.967	**152.243**	**0.578**	0.650
Bal_{33}	456.344	**219.271**	**0.944**	0.944
Bal_{34}	828.191	**363.757**	**0.750**	0.804

TABLE 11.8 – Temps d'exécution total en secondes et RMS final de g2o, Ceres et LMA avec la méthode IMPLICIT_SCHUR.

Bibliographie

[1] Hyon Lim, Jongwoo Lim, and H Jin Kim. Real-time 6-dof monocular visual slam in a large-scale environment. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 1532–1539. IEEE, 2014. XIII, 42, 44, 64

[2] Raul Mur-Artal, JMM Montiel, and Juan D Tardos. Orb-slam : a versatile and accurate monocular slam system. *arXiv preprint arXiv :1502.00956*, 2015. XIII, 42, 44, 52, 57, 64

[3] Georg Klein and David Murray. Parallel tracking and mapping for small ar workspaces. In *Mixed and Augmented Reality, 2007. ISMAR 2007. 6th IEEE and ACM International Symposium on*, pages 225–234. IEEE, 2007. XIII, 42, 44, 51, 57, 105

[4] Mohamed Tamaazousti, Vincent Gay-Bellile, SN Collette, Steve Bourgeois, and Michel Dhome. Nonlinear refinement of structure from motion reconstruction by taking advantage of a partial knowledge of the environment. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 3073–3080. IEEE, 2011. XIII, 42, 44, 106, 107, 132

[5] Dorra Larnaout, Vincent Gay-Bellile, Steve Bourgeois, Benjamin Labbé, and Michel Dhome. Fast and automatic city-scale environment modeling for an accurate 6dof vehicle localization. In *Mixed and Augmented Reality (ISMAR), 2013 IEEE International Symposium on*, pages 265–266. IEEE, 2013. XIII, 42, 44, 107

[6] Datta Ramadasan, Clément Deymier, Marc Chevaldonné, and Thierry Chateau. Slam visuel temps réel pour l'estimation précise de plan. In *Reconnaissance de Formes et Intelligence Artificielle (RFIA) 2014*, 2014. 6

[7] Datta Ramadasan, Marc Chevaldonné, and Thierry Chateau. Slam contraint en environnement de grande taille. In *Reconnaissance de Formes et Intelligence Artificielle (RFIA) 2014*, 2014. 6

[8] Datta Ramadasan, Marc Chevaldonné, and Thierry Chateau. Real-time slam for static multi-objects learning and tracking applied to augmented reality applications. In *Virtual Reality (VR) 2015*, 2015. 6

[9] Datta Ramadasan, Marc Chevaldonné, and Thierry Chateau. Dcslam : un slam temps réel à contraintes dynamiques. In *Journées francophones des jeunes chercheurs en vision par ordinateur*, 2015. 6

[10] Datta Ramadasan, Marc Chevaldonné, and Thierry Chateau. Dcslam : A dynamically constrained real-time slam. In *Image Processing (ICIP), 2015 20th IEEE International Conference on*. IEEE, 2015. 6

[11] Datta Ramadasan, Marc Chevaldonné, and Thierry Chateau. Mcslam : a multiple constrained slam. In *Proceedings of the British Machine Vision Conference (BMVC)*, pages 107.1–107.12. BMVA Press, September 2015. 6

[12] François Bardet, Thierry Chateau, Datta Ramadasan, et al. Suivi et catégorisation visuels en temps réel d'un nombre variable d'objets : application au suivi de véhicules. In *ORASIS'09-Congrès des jeunes chercheurs en vision par ordinateur*, 2009. 6

[13] François Bardet, Thierry Chateau, and Datta Ramadasan. Real-time multi-object tracking with few particles. *VISAPP, INSTICC*, pages 456–463, 2009. 6

[14] François Bardet, Thierry Chateau, and Datta Ramadasan. Unifying real-time multi-vehicle tracking and categorization. In *Intelligent Vehicles Symposium, 2009 IEEE*, pages 197–202. IEEE, 2009. 6

[15] François Bardet, Thierry Chateau, and Datta Ramadasan. Illumination aware mcmc particle filter for long-term outdoor multi-object simultaneous tracking and classification. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 1623–1630. IEEE, 2009. 6

[16] Nadir Karam, Hicham Hadj-Abdelkader, Clement Deymier, and Datta Ramadasan. Improved visual localization and navigation using proprioceptive sensors. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 4155–4160. IEEE, 2010. 6

[17] Michel Dhome, Eric Royer, Maxime Lhuilier, Datta Ramadasan, Nadir Karam, Clément Deymier, Vadim Litvinov, Hicham Hadj Abdelkader, Thierry Chateau, Jean-Mare Lavest, et al. Method of calibrating a computer-based vision system onboard a craft, October 9 2012. US Patent App. 14/350,983. 6

[18] Peter Sturm, Srikumar Ramalingam, Jean-Philippe Tardif, Simone Gasparini, and Joao Barreto. Camera models and fundamental concepts used in geometric computer vision. *Foundations and Trends® in Computer Graphics and Vision*, 6(1–2) :1–183, 2011. 10, 12

[19] Christopher Geyer and Kostas Daniilidis. A unifying theory for central panoramic systems and practical implications. In *Computer Vision—ECCV 2000*, pages 445–461. Springer, 2000. 13

[20] Camillo J Taylor and David J Kriegman. Minimization on the lie group so (3) and related manifolds. *Yale University*, 1994. 16, 155

[21] Pei Yean Lee and John B Moore. Gauss-newton-on-manifold for pose estimation. *Journal of industrial and management optimization*, 1(4) :565, 2005. 16

[22] MJ Box, William Henry Swann, and D Davies. Non-linear optimization techniques. 1969. 18

[23] Donald W Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *Journal of the Society for Industrial & Applied Mathematics*, 11(2) :431–441, 1963. 19

[24] Kenneth Levenberg. A method for the solution of certain problems in least squares. *Quarterly of applied mathematics*, 2 :164–168, 1944. 19

[25] Hans Bruun Nielsen. Damping parameter in marquardt's method. Technical report, Informatics and Mathematical Modelling, Technical University of Denmark, DTU, 1999. 19

[26] Paul R Beaudet. Rotationally invariant image operators. In *International Joint Conference on Pattern Recognition*, volume 579, page 583, 1978. 21

[27] Hans P Moravec. Rover visual obstacle avoidance. In *IJCAI*, pages 785–790, 1981. 21

[28] Chris Harris and Mike Stephens. A combined corner and edge detector. In *Alvey vision conference*. Manchester, UK, 1988. 21

[29] David G Lowe. Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, volume 2, pages 1150–1157. Ieee, 1999. 21

[30] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2) :91–110, 2004. 21, 22

[31] Eric Royer. *Cartographie 3D et localisation par vision monoculaire pour la navignation autonome d'un robot mobile*. PhD thesis, Université Blaise Pascal-Clermont-Ferrand II, 2006. 22, 47, 60

[32] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf : Speeded up robust features. In *Computer vision–ECCV 2006*, pages 404–417. Springer, 2006. 22

[33] Etienne Mouragnon. *Reconstruction 3D et localisation simultanée de caméras mobiles : une approche temps-réel par ajustement de faisceaux local.* PhD thesis, Université Blaise Pascal-Clermont-Ferrand II, 2007. 23

[34] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. Brief : Binary robust independent elementary features. *Computer Vision–ECCV 2010*, pages 778–792, 2010. 23

[35] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb : an efficient alternative to sift or surf. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2564–2571. IEEE, 2011. 23

[36] Stefan Leutenegger, Margarita Chli, and Roland Y Siegwart. Brisk : Binary robust invariant scalable keypoints. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2548–2555. IEEE, 2011. 23

[37] Martin A Fischler and Robert C Bolles. Random sample consensus : a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 1981. 26

[38] Carl De Boor. On calculating with b-splines. *Journal of Approximation Theory*, 6(1) :50–62, 1972. 28

[39] Maurice G Cox. The numerical evaluation of b-splines. *IMA Journal of Applied Mathematics*, 10(2) :134–149, 1972. 28

[40] Myoung-Jun Kim, Myung-Soo Kim, and Sung Yong Shin. A general construction scheme for unit quaternion curves with simple high order derivatives. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 369–376. ACM, 1995. 29

[41] Pierre Esterie, Joel Falcou, Mathias Gaunard, Jean-Thierry Lapresté, and Lionel Lacassagne. The numerical template toolbox : A modern c++ design for scientific computing. *Journal of Parallel and Distributed Computing*, 74(12) :3240–3253, 2014. 33

[42] T Veldhuizen and E Gannon. *Active libraries : Rethinking the roles of compilers and libraries*, volume 5. Oct, 1998. 34

[43] Krzysztof Czarnecki. Generative programming : Methods, techniques, and applications tutorial abstract. In *Software Reuse : Methods, Techniques, and Tools*, pages 351–352. Springer, 2002. 34, 91

[44] Renato F. Salas-Moreno. *Dense Semantic SLAM.* PhD thesis, Imperial College London, 2015. 41

[45] Randall C Smith and Peter Cheeseman. On the representation and estimation of spatial uncertainty. *The international journal of Robotics Research*, 5(4) :56–68, 1986. 41

[46] Philippe Moutarlier and Raja Chatila. Stochastic multisensory data fusion for mobile robot location and environment modeling. In *5th Int. Symposium on Robotics Research*, volume 1. Tokyo, 1989. 41

[47] Stephane Betge-Brezetz, Patrick Hebert, Raja Chatila, and Michel Devy. Uncertain map making in natural environments. In *Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on*, volume 2, pages 1048–1053. IEEE, 1996. 41

[48] Jose A Castellanos and Juan D Tardos. *Mobile robot localization and map building : A multisensor fusion approach.* Kluwer academic publishers, 2000. 41

[49] Andrew J Davison and David W Murray. Mobile robot localisation using active vision. In *Computer Vision—ECCV'98*, pages 809–825. Springer Berlin Heidelberg, 1998. 41

[50] Ethan Eade and Tom Drummond. Monocular slam as a graph of coalesced observations. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–8. IEEE, 2007. 41

[51] JMM Montiel, Javier Civera, and Andrew J Davison. Unified inverse depth parametrization for monocular slam. *analysis*, 9 :1, 2006. 41

[52] Paul Newman. On the structure and solution of the simultaneous localisation and map building problem. *Doctoral diss., University of Sydney*, 1999. 41

[53] Andrew J Davison. Real-time simultaneous localisation and mapping with a single camera. In *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, pages 1403–1410. IEEE, 2003. 41, 105

[54] Michael Montemerlo, Sebastian Thrun, Daphne Koller, Ben Wegbreit, et al. Fastslam : A factored solution to the simultaneous localization and mapping problem. In *AAAI/IAAI*, pages 593–598, 2002. 41

[55] Robert Sim, Pantelis Elinas, and James J Little. A study of the rao-blackwellised particle filter for efficient and accurate vision-based slam. *International Journal of Computer Vision*, 74(3) :303–318, 2007. 41

[56] Hauke Strasdat, Cyrill Stachniss, Maren Bennewitz, and Wolfram Burgard. Visual bearing-only simultaneous localization and mapping with improved feature matching. In *Autonome Mobile Systeme 2007*, pages 15–21. Springer, 2007. 41

[57] Matthew Deans and Martial Hebert. Experimental comparison of techniques for localization and mapping using a bearing-only sensor. In *Experimental Robotics VII*, pages 395–404. Springer, 2001. 42, 44

[58] Andrew W Fitzgibbon and Andrew Zisserman. Automatic camera recovery for closed or open image sequences. In *Computer Vision—ECCV'98*, pages 311–326. Springer, 1998. 42, 44

[59] Bill Triggs, Philip F McLauchlan, Richard I Hartley, and Andrew W Fitzgibbon. Bundle adjustment, a modern synthesis. In *Vision algorithms : theory and practice*, pages 298–372. Springer, 2000. 42, 44, 47, 56, 76

[60] Hauke Strasdat, José MM Montiel, and Andrew J Davison. Visual slam : why filter ? *Image and Vision Computing*, 30(2) :65–77, 2012. 42, 44

[61] Eric Royer, Maxime Lhuillier, Michel Dhome, and Jean-Marc Lavest. Monocular vision for mobile robot localization and autonomous navigation. *International Journal of Computer Vision*, 74(3) :237–260, 2007. 42

[62] Drew Steedly and Irfan Essa. Propagation of innovative information in non-linear least-squares structure from motion. In *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, volume 2, pages 223–229. IEEE, 2001. 42, 55, 127

[63] Zhengyou Zhang and Ying Shan. Incremental motion estimation through modified bundle adjustment. In *Image Processing, 2003. ICIP 2003. Proceedings. 2003 International Conference on*, volume 2, pages II–343. IEEE, 2003. 42

[64] Ron Li, Kaichang Di, Jue Wang, Sanchit Agarwal, Larry Matthies, Andrew Howard, and Reg Willson. Incremental bundle adjustment techniques using networked overhead and ground imagery for long-range autonomous mars rover localization. In *Proc. of the iSAIRA Conference*. Citeseer, 2005. 42

[65] Chris Engels, Henrik Stewénius, and David Nistér. Bundle adjustment rules. *Photogrammetric computer vision*, 2, 2006. 42

[66] Guofeng Zhang, Xueying Qin, Wei Hua, Tien-Tsin Wong, Pheng-Ann Heng, and Hujun Bao. Robust metric reconstruction from challenging video sequences. In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pages 1–8. IEEE, 2007. 42

[67] Etienne Mouragnon, Maxime Lhuillier, Michel Dhome, Fabien Dekeyser, and Patrick Sayd. Real time localization and 3d reconstruction. In *CVPR, 2006 IEEE Computer Society Conference on*, volume 1, pages 363–370. IEEE, 2006. 42, 45, 51, 76, 105, 106

[68] Gim Hee Lee, Friedrich Fraundorfer, and Marc Pollefeys. Mav visual slam with plane constraint. In *Robotics and Automation (ICRA)*. IEEE, 2011. 42

[69] Dorian Gálvez-López, Marta Salas, Juan D Tardós, and JMM Montiel. Real-time monocular object slam. *arXiv preprint arXiv :1504.02398*, 2015. 42, 106, 107, 114

[70] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics : The kitti dataset. *International Journal of Robotics Research (IJRR)*, 2013. 44, 137

[71] Georg Klein and David Murray. Parallel tracking and mapping on a camera phone. In *Mixed and Augmented Reality, 2009. ISMAR 2009. 8th IEEE International Symposium on*, pages 83–86. IEEE, 2009. 44

[72] David Nistér. An efficient solution to the five-point relative pose problem. *Pattern Analysis and Machine Intelligence*, 26(6) :756–770, 2004. 45

[73] Vincent Lui and Tom Drummond. An iterative 5-pt algorithm for fast and robust essential matrix estimation. In *British Machine Vision Conference (BMVC)*, volume 74, pages 1–11. BMVA, 2013. 45, 47, 48, 57

[74] David Nistér. An efficient solution to the five-point relative pose problem. In *CVPR (2)'03*, pages 195–202, 2003. 45

[75] Dhruv Batra, Bart Nabbe, and Martial Hebert. An alternative formulation for five point relative pose problem. In *Motion and Video Computing, 2007. WMVC'07. IEEE Workshop on*, pages 21–21. IEEE, 2007. 47

[76] Leonardo Dagum and Rameshm Enon. Openmp : an industry standard api for shared-memory programming. *Computational Science & Engineering, IEEE*, 5(1) :46–55, 1998. 47

[77] Laurent Kneip, Davide Scaramuzza, and Roland Siegwart. A novel parametrization of the perspective-three-point problem for a direct computation of absolute camera position and orientation. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 2969–2976. IEEE, 2011. 49

[78] E Malis, E Marchand, et al. Méthodes robustes d'estimation pour la vision robotique. In *Journées nationales de la recherche en robotique, JNRR'05*, 2005. 50, 57

[79] Pierre Lothe, Steve Bourgeois, Fabien Dekeyser, Eric Royer, and Michel Dhome. Towards geographical referencing of monocular slam reconstruction using 3d city models : Application to real-time accurate vision-based localization. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 2882–2889. IEEE, 2009. 51

[80] Paul A Beardsley, Andrew Zisserman, and David W Murray. Navigation using affine structure from motion. In *Computer Vision—ECCV'94*, pages 85–96. Springer, 1994. 54

[81] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*, volume 2. Cambridge Univ Press, 2000. 54, 56

[82] Richard I Hartley and Peter Sturm. Triangulation. *Computer vision and image understanding*, 68(2) :146–157, 1997. 54

[83] Karl Kraus. *Photogrammetry : V. 1, Fundamentals and Standard Processes*. Ferdinand Dummlers Verlag, 1993. 56

[84] E Mouragnon, Maxime Lhuillier, Michel Dhome, Fabien Dekeyser, and Patrick Sayd. Generic and real-time structure from motion using local bundle adjustment. *Image and Vision Computing*, 27(8) :1178–1193, 2009. 57

[85] Georg Klein and David Murray. Improving the agility of keyframe-based slam. In *Computer Vision–ECCV 2008*, pages 802–815. Springer, 2008. 57, 77, 134

[86] Pierre Lébraly. *Etalonnage de caméras à champs disjoints et reconstruction 3D : Application à un robot mobile*. PhD thesis, Université Blaise Pascal-Clermont-Ferrand II, 2012. 60, 86

[87] Edward Rosten, Reid Porter, and Tom Drummond. Faster and better : A machine learning approach to corner detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 32(1) :105–119, 2010. 64

[88] M.I. A. Lourakis and A.A. Argyros. SBA : A Software Package for Generic Sparse Bundle Adjustment. *ACM Trans. Math. Software*, 36(1) :1–30, 2009. 71, 76

[89] Rainer Kummerle, Giorgio Grisetti, Hauke Strasdat, Kurt Konolige, and Wolfram Burgard. g 2 o : A general framework for graph optimization. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 3607–3613. IEEE, 2011. 72, 76

[90] Frank Dellaert. Factor graphs and gtsam : A hands-on introduction. 2012. 72

[91] Lukáš Polok, S Viorela Ila, and Pavel Smrž. Cache efficient implementation for block matrix operations. In *Proceedings of the 21st High Performance Computing Symposia*. Association for Computing Machinery, 2013. 72, 75

[92] Lukas Polok, Viorela Ila, Marek Solony, Pavel Smrz, and Pavel Zemcik. Incremental block cholesky factorization for nonlinear least squares in robotics. In *Robotics : Science and Systems*, 2013. 72

[93] Changchang Wu, Sameer Agarwal, Brian Curless, and Steven M Seitz. Multicore bundle adjustment. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 3057–3064. IEEE, 2011. 72

[94] Sameer Agarwal and Keir Mierle. Ceres solver : Tutorial & reference. *Google Inc*, 2, 2012. 72

[95] B Jacob and G Guennebaud. Eigen is a c++ template library for linear algebra : Matrices, vectors, numerical solvers, and related algorithms, 2012. 72

[96] Tim Davis, WW Hager, and IS Duff. Suitesparse, 2013. 72

[97] Jorge Nocedal and S Wright. Numerical optimization, series in operations research and financial engineering. *Springer, New York*, 2006. 86

[98] Sameer Agarwal, Noah Snavely, Steven M Seitz, and Richard Szeliski. Bundle adjustment in the large. In *Computer Vision–ECCV 2010*, pages 29–42. Springer, 2010. 87

[99] Avanish Kushal and Sameer Agarwal. Visibility based preconditioning for bundle adjustment. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 1442–1449. IEEE, 2012. 87

[100] Yekeun Jeong, David Nister, Drew Steedly, Richard Szeliski, and In-So Kweon. Pushing the envelope of modern methods for bundle adjustment. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 34(8) :1605–1617, 2012. 87

[101] SJ Wright and John Norman Holt. An inexact levenberg-marquardt method for large sparse nonlinear least squres. *The Journal of the Australian Mathematical Society. Series B. Applied Mathematics*, 26(04) :387–403, 1985. 88

[102] Jean-Thierry Lapresté. *Introduction à MATLAB*. Ellipses, 2009. 90

[103] William Kingdon Clifford. *A preliminary sketch of biquaternions*. 1873. 90

[104] Dan Piponi. Automatic differentiation, c++ templates, and photogrammetry. *Journal of Graphics Tools*, 9(4) :41–55, 2004. 90, 91

[105] Paul Hudak. Building domain-specific embedded languages. *ACM Computing Surveys (CSUR)*, 28(4es) :196, 1996. 91

[106] George Karypis and Vipin Kumar. A high performance sparse cholesky factorization algorithm for scalable parallel computers. In *Frontiers of Massively Parallel Computation, 1995. Proceedings. Frontiers' 95., Fifth Symposium on the*, pages 140–147. IEEE, 1995. 99

[107] Richard H Byrd, Robert B Schnabel, and Gerald A Shultz. A trust region algorithm for nonlinearly constrained optimization. *SIAM Journal on Numerical Analysis*, 24(5) :1152–1170, 1987. 99

[108] Hauke Strasdat, JMM Montiel, and Andrew J Davison. Scale drift-aware large scale monocular slam. In *Robotics : Science and Systems*, volume 2, page 5, 2010. 105

[109] Maxime Lhuillier. Incremental fusion of structure-from-motion and gps using constrained bundle adjustments. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 34(12) :2489–2495, 2012. 106

[110] Hideyuki Kume, Takafumi Taketomi, Tomokazu Sato, and Naokazu Yokoya. Extrinsic camera parameter estimation using video images and gps considering gps positioning accuracy. In *Pattern Recognition (ICPR), 2010 20th International Conference on*, pages 3923–3926. IEEE, 2010. 106

[111] Julien Michot, Adrien Bartoli, and Francois Gaspard. Bi-objective bundle adjustment with application to multi-sensor slam. *3DPVT10*, 3025, 2010. 106

[112] Laurent Kneip, Margarita Chli, Roland Siegwart, Roland Yves Siegwart, and Roland Yves Siegwart. Robust real-time visual odometry with a single camera and an imu. In *BMVC*, pages 1–11, 2011. 106

[113] Michael Fleps, Elmar Mair, Oliver Ruepp, Michael Suppa, and Darius Burschka. Optimization based imu camera calibration. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 3297–3304. IEEE, 2011. 106

[114] Dorra Larnaout, Vincent Gay-Bellile, Steve Bourgeois, and Michel Dhome. Vehicle 6-dof localization based on slam constrained by gps and digital elevation model information. In *ICIP*, pages 2504–2508, 2013. 106

[115] Vadim Indelman, Stephen Williams, Michael Kaess, and Frank Dellaert. Information fusion in navigation systems via factor graph based incremental smoothing. *Robotics and Autonomous Systems*, 61(8) :721–738, 2013. 106

[116] Stefan Leutenegger, Paul Timothy Furgale, Vincent Rabaud, Margarita Chli, Kurt Konolige, and Roland Siegwart. Keyframe-based visual-inertial slam using nonlinear optimization. In *Robotics : Science and Systems*, 2013. 106, 107

[117] Paul Furgale, Timothy D Barfoot, and Gabe Sibley. Continuous-time batch estimation using temporal basis functions. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 2088–2095. IEEE, 2012. 106, 107, 123

[118] Steven Lovegrove, Alonso Patron-Perez, and Gabe Sibley. Spline fusion : A continuous-time representation for visual-inertial fusion with application to rolling shutter cameras. In *Proceedings of the British machine vision conference*, pages 93–1, 2013. 106, 107, 123

[119] Olivier A Bauchau and Lorenzo Trainelli. The vectorial parameterization of rotation. *Nonlinear dynamics*, 32(1) :71–92, 2003. 106

[120] Sid Yingze Bao, Manmohan Chandraker, Yuanqing Lin, and Silvio Savarese. Dense object reconstruction with semantic priors. In *CVPR*, pages 1264–1271. IEEE, 2013. 106, 107

[121] Tommi Tykkala, Andrew I Comport, and Joni-Kristian Kamarainen. Photorealistic 3d mapping of indoors by rgb-d scanning process. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 1050–1055. IEEE, 2013. 106, 107

[122] Renato F Salas-Moreno, Richard A Newcombe, Hauke Strasdat, Paul HJ Kelly, and Andrew J Davison. Slam++ : simultaneous localisation and mapping at the level of objects. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pages 1352–1359. IEEE, 2013. 106, 107

[123] Yang Chen and Gérard Medioni. Object modelling by registration of multiple range images. *Image and vision computing*, 10(3) :145–155, 1992. 106, 107

[124] Yuichi Taguchi, Yong-Dian Jian, Srikumar Ramalingam, and Chen Feng. Point-plane slam for hand-held 3d sensors. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 5182–5189. IEEE, 2013. 106, 107

[125] Amaury Dame, Victor A Prisacariu, Carl Y Ren, and Ian Reid. Dense reconstruction using 3d object shape priors. In *CVPR*, pages 1288–1295. IEEE, 2013. 106, 107

[126] Gerhard Reitmayr, Ethan Eade, and Tom W Drummond. Semi-automatic annotations in unknown environments. In *Mixed and Augmented Reality, 2007. ISMAR 2007. 6th IEEE and ACM International Symposium on*, pages 67–70. IEEE, 2007. 107

[127] Geraldo Silveira, Ezio Malis, and Patrick Rives. The efficient e-3d visual servoing. *International Journal of Optomechatronics*, 2(3) :166–184, 2008. 107

[128] John Bastian, Ben Ward, Rhys Hill, Anton van den Hengel, and Anthony Dick. Interactive modelling for ar applications. In *Mixed and Augmented Reality (ISMAR), 2010 9th IEEE International Symposium on*, pages 199–205. IEEE, 2010. 107

[129] Pierre Lothe, Steve Bourgeois, Eric Royer, Michel Dhome, and Sylvie Naudet-Collette. Real-time vehicle global localisation with a single camera in dense urban areas : Exploitation of coarse 3d city models. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 863–870. IEEE, 2010. 113

[130] Mohamed Tamaazousti, Vincent Gay-Bellile, Sylvie Naudet Collette, Steve Bourgeois, and Michel Dhome. Real-time accurate localization in a partially known environment : Application to augmented reality on textureless 3d objects. In *The 2nd International Workshop on AR/MR Registration, Tracking and Benchmarking*, 2011. 113, 114

[131] William H Press. *Numerical recipes in Fortran 77 : the art of scientific computing*, volume 1. Cambridge university press, 1992. 116

[132] Sunil Arya, David M. Mount, Nathan S. Netanyahu, Ruth Silverman, and Angela Y. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *J. ACM*, 45(6) :891–923, November 1998. 118

[133] Marc Chevaldonné, Marc Neveu, Frédéric Mérienne, Michel Dureigne, Nicolas Chevassus, and François Guillaume. Digital mock-up database simplification with the help of view and application dependent criteria for industrial virtual reality application. In *Proceedings of the Tenth Eurographics conference on Virtual Environments*, pages 113–122. Eurographics Association, 2004. 118

[134] Angélique Loesch, Steve Bourgeois, Vincent Gay-Bellile, and Michel Dhome. Localisation temps-réel d'objets complexes. In *Journées francophones des jeunes chercheurs en vision par ordinateur*, 2015. 134